University of Ljubljana
Faculty of computer and information science

Petrovska Ana

An embedded system for access control with a Bluetooth module

bachelor's thesis

undergraduate university study programme
computer and information science

Thesis supervisor: assoc. prof. dr. Patricio Bulić

Ljubljana 2014

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Petrovska Ana

Vgrajen sistem za kontrolo dostopa z modulom Bluetooth

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana 2014

The text is formatted with the editor LaTeX.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

The Faculty of Computer and Information Science issues the following thesis:

Develop an embedded system for access control, which is based on the STM32F407 system-on-chip. User identification should be based on the RFID tags. The system should record each access and send the records to an Android-based device via Bluetooth. Firmware for the embedded system should be based on the FreeRTOS operating system.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Implementirajte vgrajen sistem za kontrolo dostopa, ki temelji na sistemu na čipu STM32F4. Prepoznavanje uporabnikov naj temelji na značkah RFID. Sistem naj vsak dostop zabeleži in pošlje na napravo s sistemom Android preko povezave Bluetooth. Vgradna programska oprema (firmware) naj temelji na operacijskem sistemu za delo v realnem času FreeRTOS.

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Ana Petrovska, z vpisno številko **63110394**, sem avtor diplomskega dela z naslovom:

Vgrajen sistem za kontrolo dostopa z modulom Bluetooth

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricia Bulića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 8. septembra 2014 Podpis avtorja:

First, I would like to express my gratitude to my mentor prof. dr. Patricio Bulić and to assistant Rok Češnovar, for all the assistance, help and useful advice that I received during the preparation of my bachelor.

Thanks to Tilen, and all the other colleagues who somehow have helped me during my studies.

Also, I want to thank Sirma, who is more than a best friend, for all the moments that we have spent together in these three years during our studies, for all the help and support, and for all the moments in the last seven years in which we constantly learned and repeated the lessons what a true friend means.

At the end, I want to express the biggest thanks to my mother and my father, for being the most patient, the most supportive and the most caring parents in the whole world. Thank you for believing in me, even in the times when I didn't believe in myself.

За мојот татко.

Contents

List of abbreviations

abbreviation	in English
TFT	Thin Film Transistors
LCD	Liquid-Crystal Display
IDE	Integrated Development Environment
MCU	Microcontroller
API	Application Program Interface
FPU	Floating-Point Unit
DSP	Digital Signal Processor
APB	Advanced Peripheral Bus
AHB	ARM added AMBA High-performance Bus
MPU	Memory Protection Unit
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
RTC	Real-Time Clock
PWM	Pulse-Width Modulation
RNG	Random Number Generator
SPI	Serial Peripheral Interface
I2C	Inter-IC Bus
PLL	Phase-Locked Loop
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

abbreviation	**in English**
OTG	USB On-The-Go
ULPI	UTMI + Low Pin Interface
CAN	Controller Area Network
SDIO	Secure Digital Input Output
MMC	Microsoft Management Console
FSMC	Flexible Static Memory Control
PLC	Programmable Logic Controller
HVAC	Heating Ventilation Air Conditioning
SPP	Serial Port Protocol
CSR	Control/Status Register
CMOS	Complementary Metal–Oxide–Semiconductor
AFH	Adaptive Frequency Hopping Feature
EDR	Enhanced Data Rate
PIO	Programmed Input/Output
RAM	Random Access Memory
GDDRAM	Graphic Display Data RAM
MIDP	Mobile Information Device Profile
OS	Operating System
CPU	Central Processing Unit
GPU	Graphics Processing Unit
Cortex-M4F	Cortex-M4 core with FPU
ADT	Android Developer Tools
SDK	Software Development Kit
UUID	Universally Unique Identifier
GPIO	General Purpose Input/Output
NVIC	Nested Vector Interrupt Controller
TX	Transmit Data Output
RX	Receive Data Input

Abstract

Security and privacy issues have remained a challenge for years. Because of that it is very important to have your own system for access control, which is under your control, in every moment, when you are having your mobile device with you.

In this work, we will present an embedded system for access control based on STM32F4 discovery board. The discovery board is based on STM32F407VGT6 microcontroller, with ARM® Cortex™-M4F core with embedded Flash and SRAM. On the board are connected Bluetooth HC-05 module, RFID reader and TFT Touchscreen module. User authentication in the embedded system is based on the RFID technology. With every authentication the name of the user will be shown on the TFT Touchscreen module, and at the same time it will be sent via Bluetooth to the Android application. The string is being sent over UART.

Keywords: access control, Android application, Bluetooth, RFID technology

Povzetek

Vprašanje varnosti in zasebnosti je izziv že leta. Zaradi tega je pomembno imeti lasten system za nadzor dostopa, ki ga bo uporabnik nadzoroval ob vsakem trenutku, ko ima s seboj svojo mobilno napravo.

V tem delu bomo predstavili vgrajen system za nadzor dostopa, temelječ na STM32F4 razvojni plošči. Razvojna plošča temelji na mikrokrmilniku STM32F407VGT6 z jedrom ARM® Cortex™-M4F in vgrajenim pomnilnikom Flash in SRAM. Na plošči so povezani Bluetooth modul HC-05, bralnik RFID in modul z TFT zaslonom na dotik. Avtentikacija uporabnikov v tem sistemu temelji na tehnologiji RFID. Z vsako avtentikacijo se bo na zaslonu TFT izpisalo ime uporabnika, hkrati pa bo preko Bluetootha poslano k Android aplikaciji. Niz znakov bo poslan preko UART-ja.

Ključne besede: kontrolo dostopa, Android aplikacija, Bluetooth, RFID tehnologija

Chapter 1

Introduction

1.1 Introduction

Mobile phones have become a necessity of life. Today we are use mobile phones in our everyday life for many different purposes, firstly as a phone, but as technology improves during the years, the main, the primary purpose of mobile phones, loses its value. With the advances of technology, mobile phones are becoming a way of life. For many people, mobile phones are fundamental in organizing and controlling their lives. The mobile phones of today have become an essential utility tool and with technological advancements, the mobile phones today perform many functions in tune with the modern life style. Today's improved mobile phones, called smartphones, have so many features that in the past, we would have found only on a personal digital assistant or a computer. In general, a smartphone is based on an operating system that allows it to run applications. Today we are able to download a variety of applications, such as personal and business finance managers, handy personal assistants, or applications for almost anything. In addition, one of the most important aspects in every person life is improving its own security, and smartphones can really help and find an easy and smart solution.

Security and privacy issues have remained a challenge for years. People through the years, have always been seeking to protect their territory against unwanted visitors. An access control system is the selective restriction of access to a place or other resource [1]. The act of accessing may mean consuming, entering, or

using. The term access control refers to the practice of restricting entrance to a property, a building, or a room to authorized persons. Physical access control can be achieved by a human, through mechanical means such as locks and keys, or through technological means such as access control systems. Electronic access control uses computers to solve the limitations of mechanical locks and keys. A wide range of credentials can be used to replace mechanical keys. When a credential is presented to a reader, the reader sends the credential's information, usually a number, to a control panel, a highly reliable processor. The control panel compares the credential's number to an access control list, grants or denies the presented request, and sends a transaction log to a database. When access is denied based on the access control list, the door remains locked. If there is a match between the credential and the access control list, the control panel operates a relay that in turn unlocks the door.

The purpose of the thesis is to develop a system for access control, which is based on RFID (Radio Frequency Identification), and exactly that system to be controlled by the Android application. The RFID tags will be used as credentials, and the RFID antenna will be the reader. When a specific user will be recognized, his/her name will be written on a specific screen, and at the same moment the user's name will be sent via Bluetooth module, to the mobile device, and it will be shown also in the Android application. With that, the person who uses the application has control over the system for access control.

In the first part of this thesis we will have a quick review of all the technologies that we will use for building the embedded system and the Android application, and how do they work. Also in this part, we will have an overview of the software development environment CoIDE, used for building the embedded system.

In the second part, we will deeply explain our embedded system for access control and the Android application. First, we will explain the hardware design, and after that the software solution.

Chapter 2

Theoretical background

2.1 Bluetooth

Bluetooth is a wireless technology that is embedded in the electronic gadgets. It allows you to talk and share information such as music, voice and video wirelessly. Bluetooth technology uses radio waves just like mobile phones, TV and FM radio. The difference between these devices and Bluetooth wireless technologies is the distance. Radio and television broadcast receivers, too many people in many miles. Bluetooth technology only sends information to your personal area. This space is called the personal PAN Personal Area Network.

The technology used by Bluetooth turns was discovered in 1940 pioneers by the military. Engineers in a Swedish company called Ericsson Bluetooth technology invented in 1994. A group of companies in 1998, worked together to connect their products using Bluetooth technology. Seeing that the technology worked, companies formed the special interest group (SIG) Bluetooth. This organization is dedicated to keeping this technology [2].

The Bluetooth RF transceiver (or physical layer) operates in the unlicensed ISM band centered at 2.4 gigahertz (the same range of frequencies used by microwaves and Wi-Fi). The core system employs a frequency-hopping transceiver to combat interference and fading [3]. Bluetooth devices are managed using an RF topology known as a star topology. A group of devices synchronized in this fashion forms a piconet, that is shown in Figure 2.1, which may contain one master and up to

seven active slaves, with additional slaves that are not actively participating in the network. (A given device may also be part of one or more piconets, either as a master or as a slave.) In a piconet, the physical radio channel is shared by a group of devices that are synchronized to a common clock and frequency-hopping pattern, with the master device providing the synchronization references.

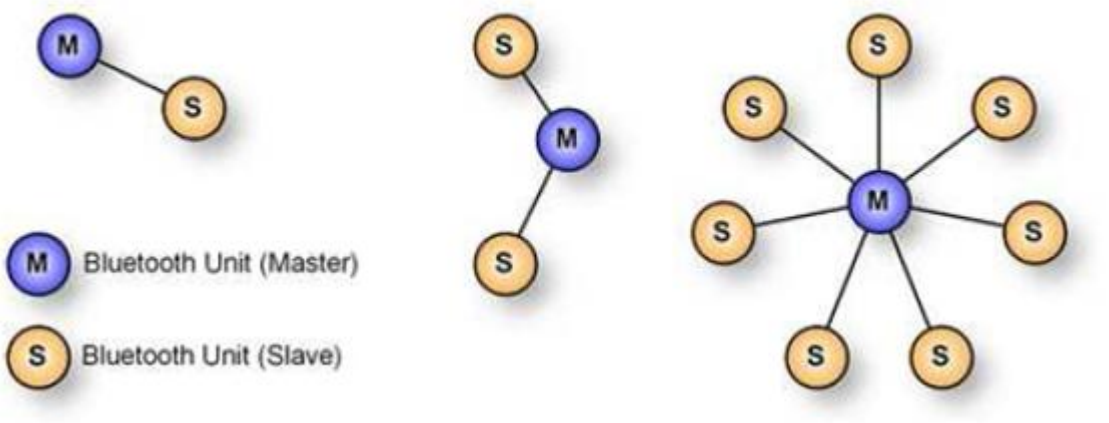


Figure 2.1: Bluetooth Piconet

Let's say the master device is your mobile phone. All of the other devices in your piconet are known as slaves. This could include your headset, GPS receiver, MP3 player, car stereo, and so on.

2.1.1 Service Discovery Protocol

The Service Discovery Protocol (SDP) allows a device to discover services offered by other devices, and their associated parameters.

For example, when you use a mobile phone with a Bluetooth headset, the phone uses SDP to determine which Bluetooth profiles the headset can use (Headset Profile, Hands Free Profile, Advanced Audio Distribution Profile (A2DP) etc.) and the protocol multiplexer settings needed for the phone to connect to the headset using each of them. Each service is identified by a Universally Unique Identifier (UUID), with official services (Bluetooth profiles) assigned a short form UUID (16 bits rather than the full 128) [4].

2.2 RFID

The roots of radio frequency identification technology can be traced back to World War II. The Germans, Japanese, Americans and British were all using radar to warn of approaching planes while they were still miles away. The problem was there was no way to identify which planes belonged to the enemy and which were a country's own pilots returning from a mission.

The Germans discovered that if pilots rolled their planes as they returned to base, it would change the radio signal reflected back. This crude method alerted the radar crew on the ground that these were German planes and not Allied aircraft (this is, essentially, the first passive RFID system).

It was during the 1960s that RFID was first considered as a solution for the commercial world. The first commercial applications involving RFID followed during the 70s and 80s. These commercial applications were concerned with identifying some asset inside a single location. They were based on proprietary infrastructures. The third era of RFID started in 1998, when researchers at the Massachusetts Institute of Technology (MIT) Auto-ID Center began to research new ways to track and identify objects as they moved between physical locations. This research, which has a global outlook, centered on radio frequency technology and how information that is held on tags can be effectively scanned and shared in near real time [5].

An RFID system is composed of three main components:

- tags
- a reader and its antennas
- a middleware application that is integrated into a host system

2.2.1 RFID tags

RFID tags, also known as transponders, contain a chip and an antenna. The latter enables the chip to respond to an interrogation signal transmitted from the RFID reader. RFID tags can be attached to or embedded in a physical object to be identified. They store product-item information such as manufacturer, product lot, size and category, production date, expiration date, final destination, etc. RFID tags have various characteristics such as designs, power source, carrier frequency,

communication method, read range, data storage capacity, memory type, size, operational life, and cost (Figure 2.2). For instance, the tags could be

- read only, write once/read many times or read/write capable
- active, passive or semi-active depending on how the operating power is driven.

Active RFID and Passive RFID are fundamentally different technologies that are often evaluated together. While both use radio frequency energy to communicate between a tag and a reader, the method of powering the tags is different.

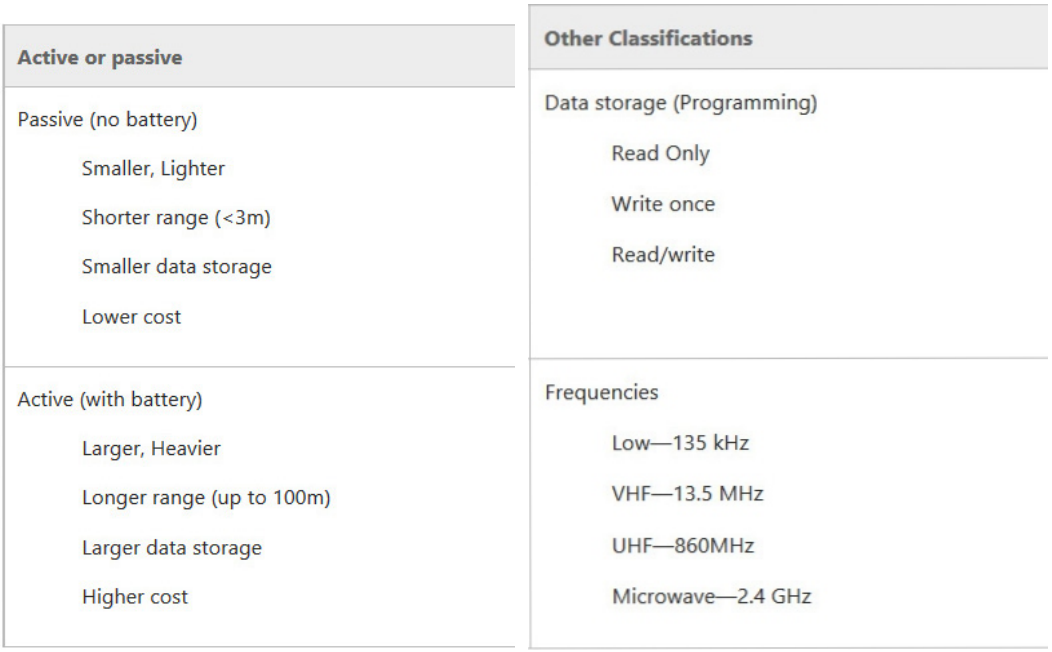
Active RFID uses an internal power source (battery) within the tag to continuously power the tag and its RF communication circuitry, whereas Passive RFID relies on RF energy transferred from the reader to the tag to power the tag. Passive RFID requires stronger signals from the reader, and the signal strength returned from the tag is constrained to very low levels. Active RFID allows very low power RF beacons to be received by the reader because the reader does not need to power the tag for the tag to respond. Additionally, the Active RFID tag is continuously powered, whether in the vicinity of the reader or not. Active tags can also "beacon" or initiate communication with a reader (or other tags) when certain conditions are present. Active tags can contain external sensors to monitor temperature, humidity, motion, and other conditions [6].

2.3 Thin Film Transistor touch screen

A TFT touch screen is a combination device that includes a TFT LCD display and a touch technology overlay on the screen. The device can both display content and act as an interface device for whoever is using it.

2.3.1 About TFT Screens

Thin Film Transistors (TFTs), also called TFT screens, are a type of active matrix LCD display capable of displaying millions of high-contrast, clear and bright color pixels. TFTs are used in HDTV sets, computer monitors, laptop monitors, tablets, personal media players, smartphones and even feature phones. The first TFT



Active or passive
Passive (no battery)
Smaller, Lighter
Shorter range (<3m)
Smaller data storage
Lower cost
Active (with battery)
Larger, Heavier
Longer range (up to 100m)
Larger data storage
Higher cost

Other Classifications
Data storage (Programming)
Read Only
Write once
Read/write
Frequencies
Low—135 kHz
VHF—13.5 MHz
UHF—860MHz
Microwave—2.4 GHz

(a) Active or passive (b) Other Classifications

Figure 2.2: RFID Tags Types

screen came with the IBM ThinkPad's 1992 model. The TFT technology works by controlling brightness in red, green and blue sub-pixels through transistors for each pixel on the screen. The pixels themselves do not produce light; instead, the screen uses a backlight for illumination. The TFT family also includes LEDs, which are a type of LCD screen that uses an LED as a backlight.

2.3.2 About Touch Screens

Touch screens are a type of overlay placed on a display screen used to register touch interaction on the screen. Touch screens are not a type of display, but rather a component that can be added to an existing screen. Touch screens use two different methods to register touch interaction called "resistive" and "capacitive," which refer to pressure and touch sensitivity respectively. Resistive screens work by measuring voltage differences caused by finger or stylus pressure on the screen, whereas capacitive screens work by measuring current interruption.

Resistive Touchscreens

A resistive touchscreen is made out of two thin layers separated by a thin gap. These are not the only layers in the resistive touchscreen, but we'll focus on them for simplicity. These two layers both have a coating on one side, with the coated sides facing each other inside the gap. When these two layers of coating touch each other, a voltage is passed, which is in turn processed as a touch in that location.

So when your finger, stylus, or any other instrument touches a resistive screen, it creates a slight pressure on the top layer, which is then transferred to the adjacent layer, thus starting the cascade of signals. Because of this, you can use anything you want on a resistive touchscreen to make the touch interface work; a gloved finger, a wooden rod, a fingernail – anything that creates enough pressure on the point of impact will activate the mechanism and the touch will be registered.

For this very same reason, resistive touchscreen require slight pressure in order to register the touch, and are not always as quick to respond as capacitive touchscreens such as other devices, for example the iPhone's. In addition, the resistive touchscreen's multiple layers cause the display to be less sharp, with lower contrast than we might see on capacitive screens. While most resistive screens don't allow for multi-touch gestures such as pinch to zoom, they can register a touch by one finger when another finger is already touching a different location on the screen [7].

2.3.3 Combined powers of TFT and touch screens

TFT touch screens use both the TFT and touch screen technologies together to create a touch-based interface overlay on a thin, lightweight display. Touch screens were in use before TFT LCD dominated the large display market and formerly existed as overlays for CRT screens. However, CRT screens require a much larger footprint and weigh several times more for the same screen space as LCD screens. Combining touch screens with TFT displays helped make the concept practical and affordable.

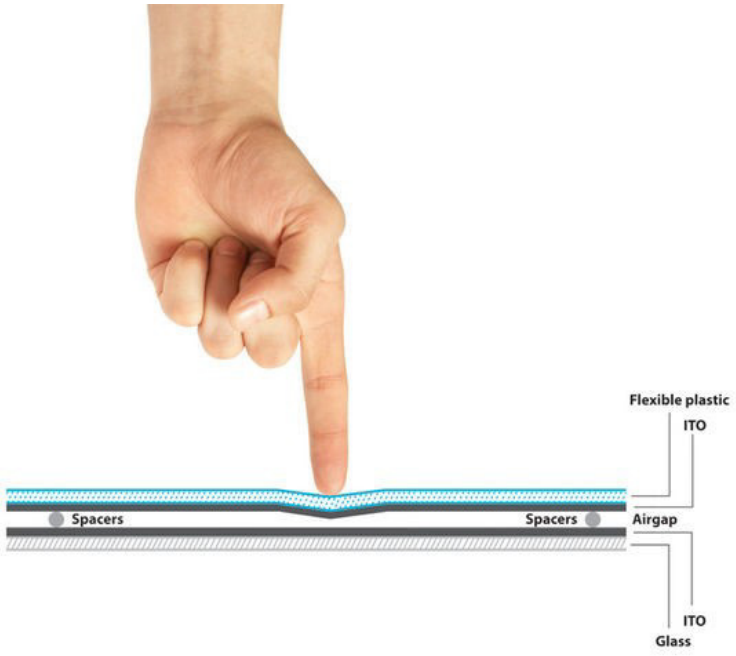


Figure 2.3: Resistive touchscreen

2.4 CooCox

2.4.1 Overview

Cooperate on Cortex - CooCox is a Free and Open ARM development tool environment for ARM Cortex-M4, M3 and M0 microcontrollers from. CooCox offer freely available powerful software development tool for ARM Cortex-M based microcontrollers. CooCox Tools organizes and provides the knowledge required by developers. It's also a social network platform for embedded developers and provides a embedded development knowledge platform to the users. CooCox can organize information, extract and share expertise through the collective wisdom.

CooCox development tool consist of various modules like CoIDE, CoFlash, CoLinkEx, CoOS, CoCenter and CoAssistant, Figure 2.4.

2.4.2 CoIDE

CoIDE includes all the tools necessary to develop high-quality software solutions in a timly and cost effective manner. It integrates CoBuilder and CoDebugger

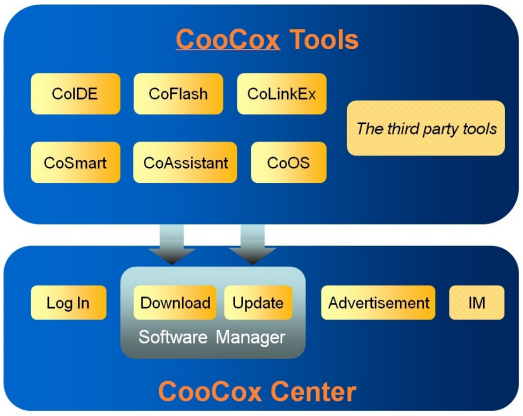


Figure 2.4: CooCox Modules

for simplicity and ease of use. CoIDE is consists of a component-based network platform and an Eclipse-based development environment [Component is a set of relatively independent codes, peripheral driver programs, some algorithms and operating system]. Unlike other traditional embedded development IDE, CooCox programming approach is based on its components, routines and examples. The component-based network platform can quickly obtain resources to build applications, share your resources and exchange ideas. The Eclipse-based development environment provides many features for code editing, code browsing, compiling, linking, debugging, project management, etc.

CoIDE has four different panels windows like any other IDEs but serves different purpose: Components Window, Components Tree Window, Example Window and Help Window [8]. They are shown in Figure 2.5.

Components Window:

Components window is basically a browser window where you can select your MCU supplier and the microcontroller. After selecting your microcontroller, here you can see the listed of all the components for respective microcontroller. User can add the appropriate component code to their project only by clicking on the components. If the components are appropriate to user requirement, user only need to check the corresponding components to complete all of the software development work except the application layer development.

Any registered users can upload their own components to the server following

CooCox component standard and then share them with other users. Developers are able to offer comments and grade the components uploaded by other users, components window will rank them by these scores.

Components Tree Window:

The checked components will be organized in the form of tree in Components Tree Window, which is shown in Figure 2.5 with the red colored frame. The Component Tree is completely corresponding to the Project tree. When checking one component, the corresponding files in Project tree will be highlighted, shown in Figure 2.5 with the yellow color. And the Examples Window will automatically display the example codes for the selected component.

Examples Window:

Once clicked on any component in Components Tree Window, all the related examples will be listed in Examples Window. Examples Window is shown in Figure 2.5 with the blue colored frame. User can refer to these examples or add them to their project. Like Components Window, Examples Window is also a browser based and registered user can add their own examples to share with others or give comments and grade to those provided by other users.

Help Window:

Help Window that is shown in Figure 2.5 with the green color, provides a variety of related information according to the changes made by the developer. When user selects a microcontroller, Help Window will provide its basic information, data sheet, technical manuals. Checking on component, Help Window will provide its API information. Checking for examples, help window will provide its schematic and evaluation board information.

2.4.3 Features for CoIDE

- Free to use
- Full functional IDE

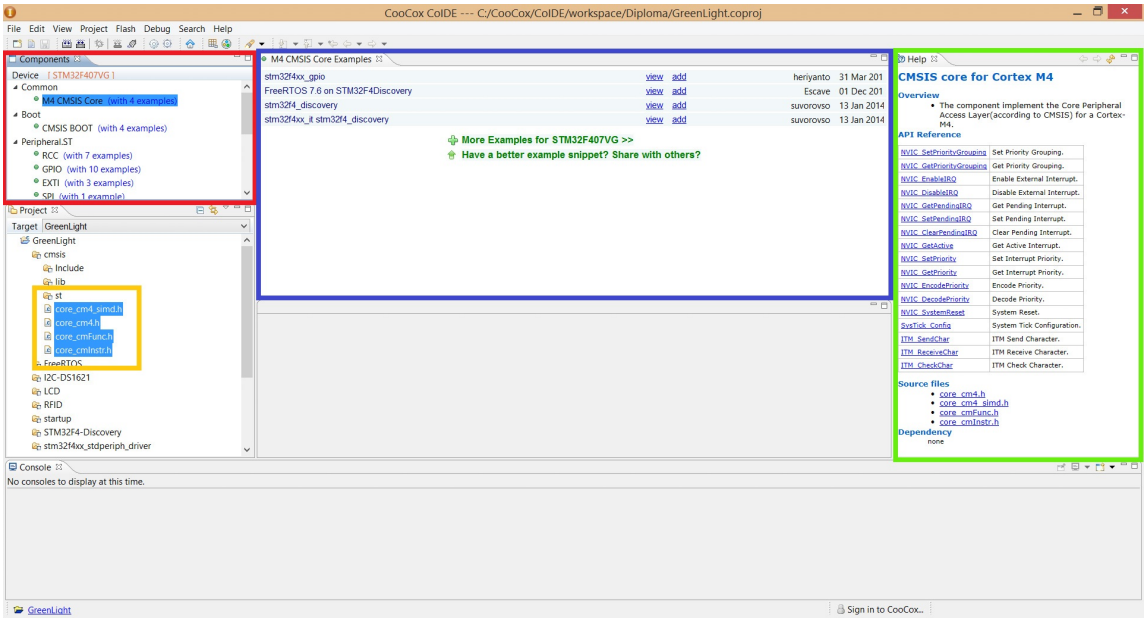


Figure 2.5: CooCox Windows

- Component-oriented development platform
- Internet-based, efficient integration of network resources
- Integrates CoOS
- Peripheral registers

2.4.4 Creating project in CoIDE

CoIDE is purely free to use for both non-commercial and commercial applications with no code size limit. Although the IDE itself is not open-source, the code components available in CoIDE are free and open to use. Code components (sometimes simply called "components"), in CooCox community, refer to reusable code snippets which are relatively independent from each other, including boot code, peripheral libraries, drivers, OS, middle layer software, examples, etc. Code components are presented in Figure 2.6.

When we launch CoIDE, we will see this Welcome page that displays, shown in Figure 2.7. In the Welcome page, we have quick access to the code component Repository, we have access to create a new project, as well as open an existing

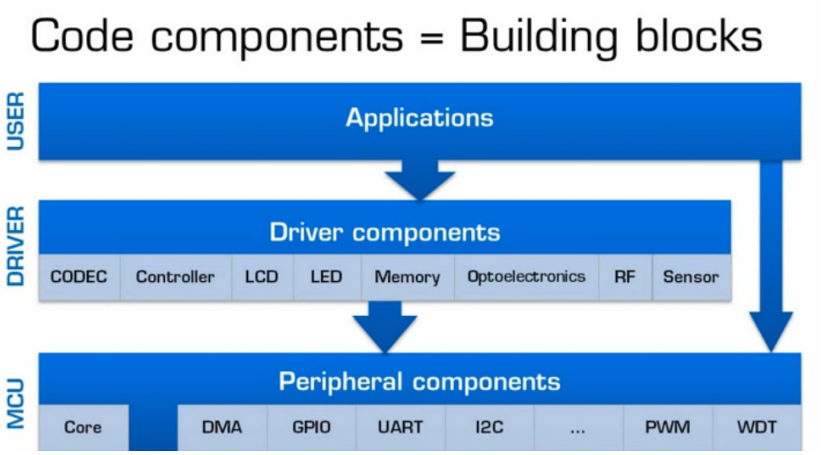


Figure 2.6: CooCox Code Components

CoIDE project. We also have link to CoIDE User Guide, as well as CooCox Forum for communication and free technical support.

For creating a new project we select "Create a New Project" from the Welcome page, and we will see a wizard window pop up. We have to enter a project name, after that we have to select the model. We can create a new project either based on our target chip or based on our target board. The wizard pop up window is shown in Figure 2.8. We select "Board". After that in the search text box we select the preferred discovery board. We can see from the window the overview of the board that we selected. When we click "Finish" button, CoIDE will automatically generate our project, with a main.c file.

The Repository view, shown in Figure 2.9, shows basic information of the board including the purchase linkage and document in the "Board" page. We can view all target boards CoIDE supports from the "Board List" page. If we click the "Document" button, the Help will show the document of the board. It also lists all the driver components and examples available for the target board.

The "Peripherals" page, shown in Figure 2.10, lists all peripherals components available for the target chip, and the "Drivers" page, shown in Figure 2.11, lists all driver components available in CooCox component database.

Each page has "Refresh" and "Upload" buttons, shown on the top right corner in the Figure 2.11, to help you get latest components and share with others.

If we click on a component in the Repository view, as a result the Help view will

Figure 2.7: CoIDE Welcome Page

show document of the component. We can add source code of a component to our project by checking the check box in the front. We can check all the component which we want to use in our project, and the source code will be added to the project automatically at the same time.

We can click on "Configuration" button to open the Configuration view, shown in Figure 2.12. We can do many thing in this view, like changing target device, configure optimization options, use our own linker script, specify output file format, use Before/After Build functions, specify our own programming algorithm, etc. CoIDE supports a number of debug adapters including ST-Link.

After configuration, we can compile our project via clicking the "Build" button and view compiled information in the Console window. After that we have to click

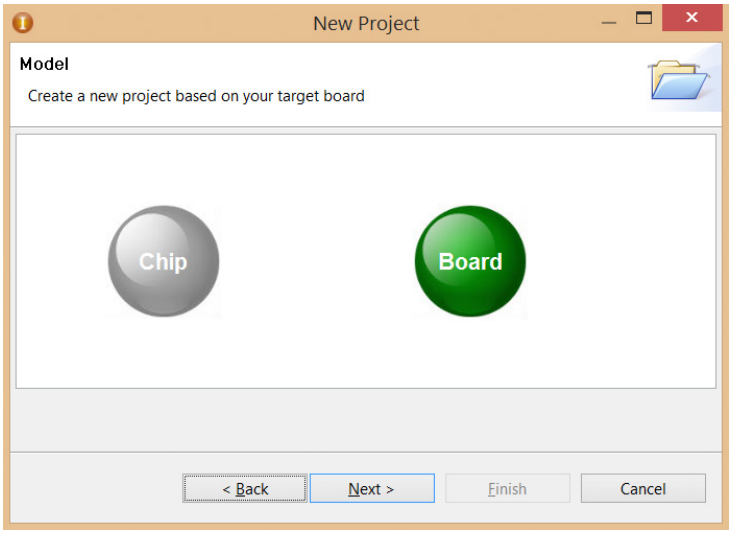


Figure 2.8: Wizard Pop Up Window

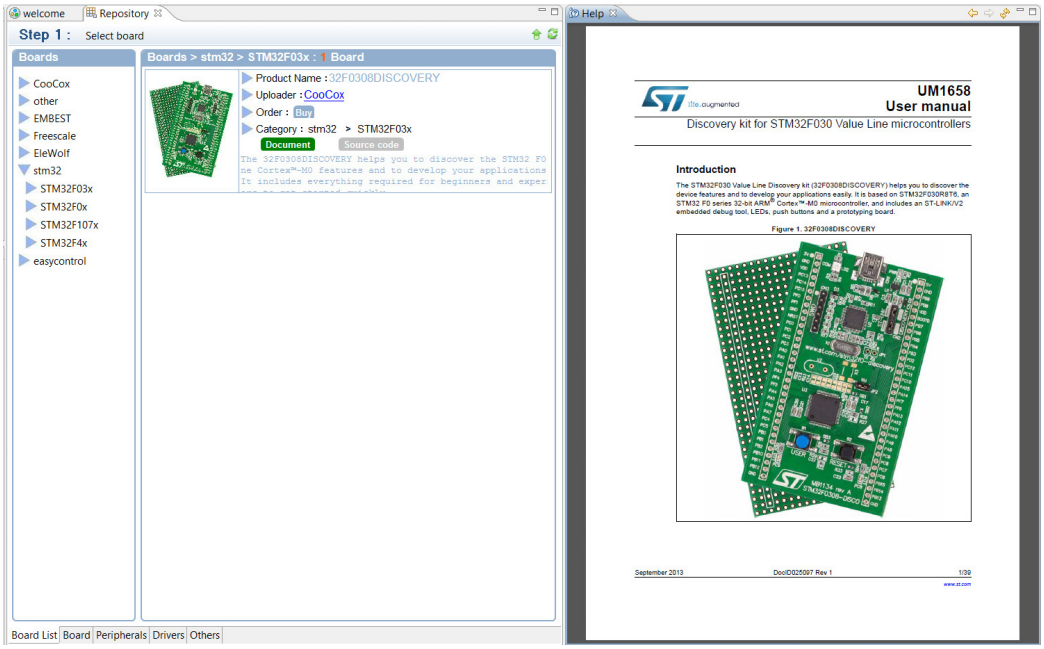


Figure 2.9: Repository View

"Download" button to download the project to our specified Discovery Board. If we want to debug, we have to click the "Debug" button to start debugging. The "Build" button, "Debug" button and "Download" button are presented in Figure 2.13, with

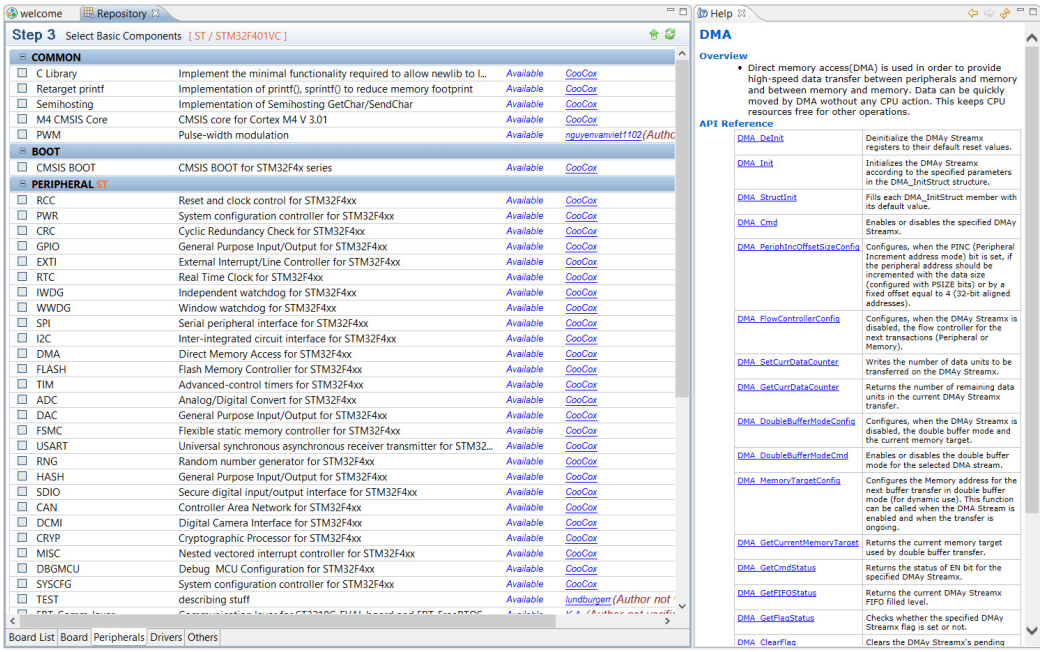


Figure 2.10: Peripherals Page

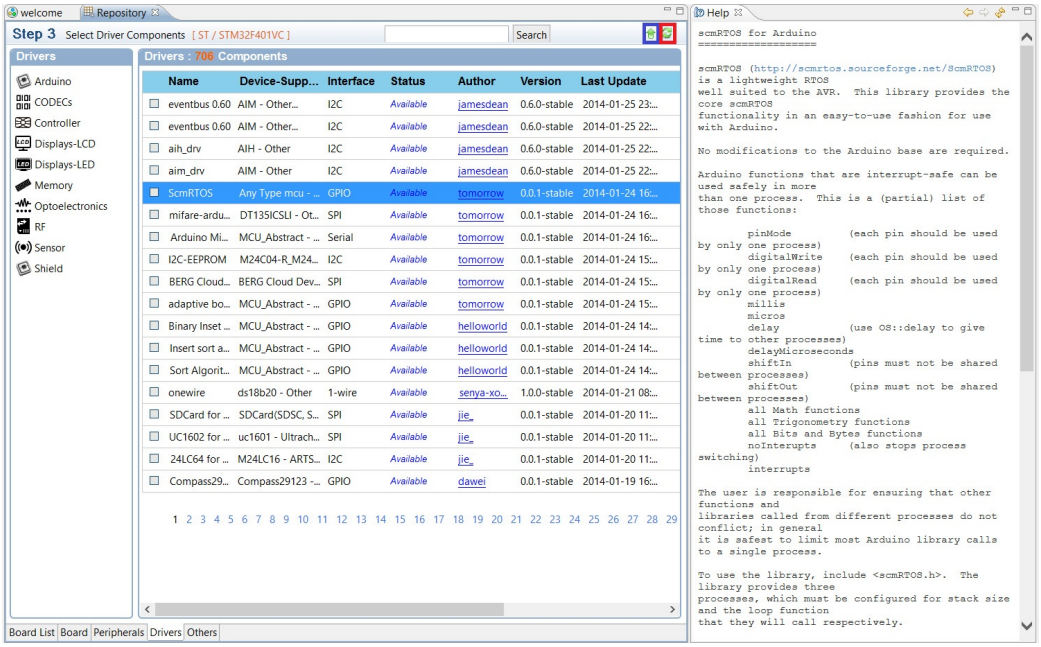


Figure 2.11: Drivers Page

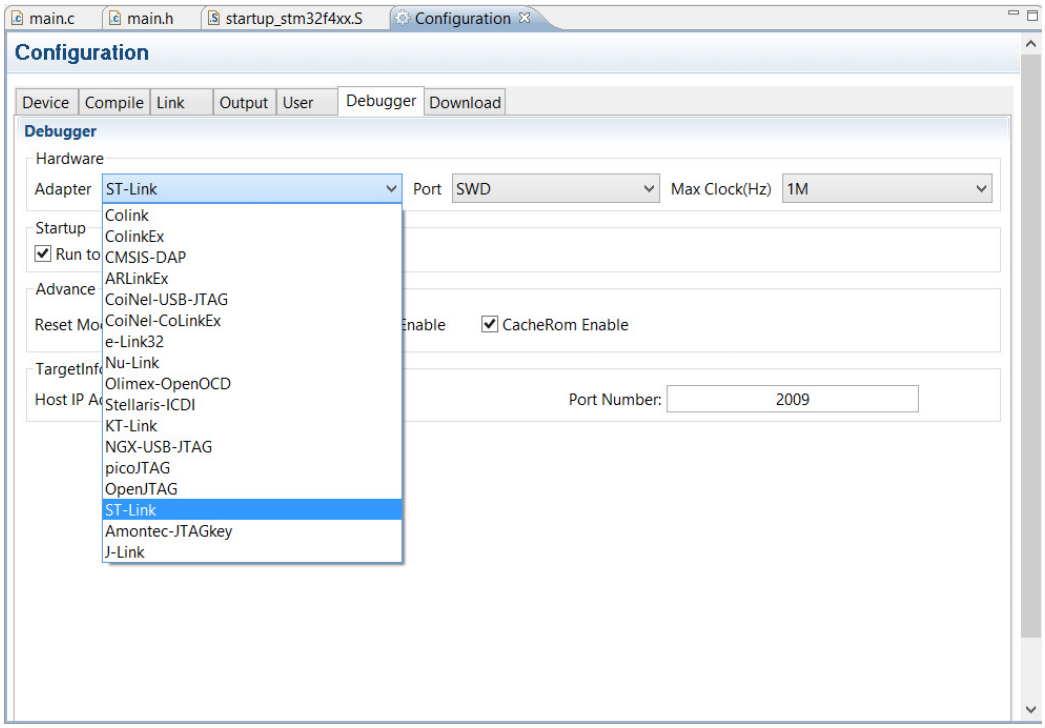


Figure 2.12: Configuration View

red, orange and blue color, respectively.

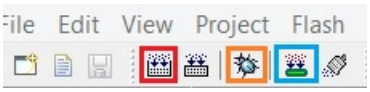


Figure 2.13: "Build", "Debug" and "Download" Buttons

From the view menu, you can open Registers view, Peripherals view, Disassembly view, Variables view, Memory view, etc. All the views are shown in Figure 2.14, Peripherals view in red, Disassembly view in yellow, Registers view in green and Variables view in blue. The change of peripheral register values can been seen from the Peripherals view, the change of the core register values from the Registers view, and the change of the variable values from the Variables view.

We can double click to set breakpoints, and skip all breakpoints via the Breakpoints window.

If we want to resume execution we have to click on the "Run" button, to halt current execution we have to click "Suspend" button, to reset the debugging – "Reset

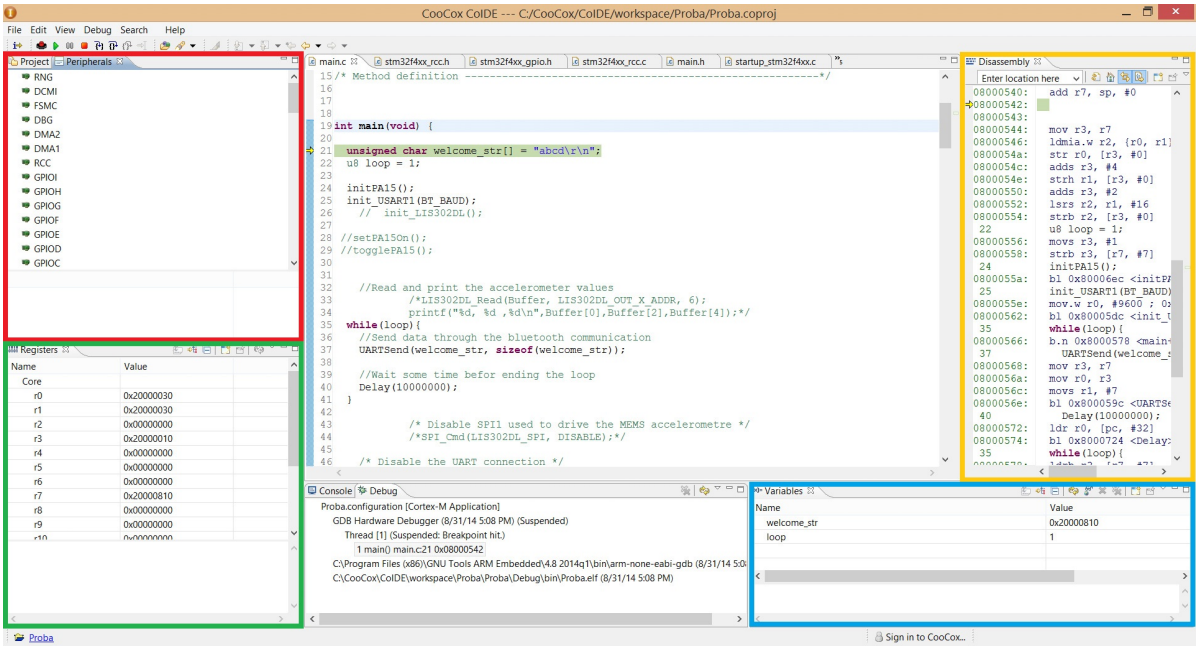


Figure 2.14: Peripherals, Disassembly, Registers and Variables Views

CPU" and "Terminate" button to ends the debug session. In the Figure 2.15 "Reset CPU" is presented with blue color, "Run" button with orange, "Suspend" button with green and "Terminate" button with red.

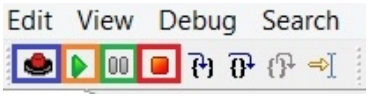


Figure 2.15: Reset CPU, Run, Suspend and Terminate buttons

2.4.5 Android operating system

Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. With a user interface based on direct manipulation, Android is designed primarily for touchscreen mobile devices such as smartphones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear). The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touchscreen input, it also has been used in game consoles, digital cameras, and other electronics.

Android is popular with technology companies which require a ready-made, low-cost and customizable operating system for high-tech devices. Android's open nature has encouraged a large community of developers and enthusiasts to use the open-source code as a foundation for community-driven projects, which add new features for advanced users or bring Android to devices which were officially released running other operating systems. The operating system's success has made it a target for patent litigation as part of the so-called "smartphone wars" between technology companies [9].

Chapter 3

Embedded system for access control controlled by an Android application

3.1 Hardware design

For creating our CoIDE project, we used STM32F4 Discovery Board, based on STM32F407VGT6 microcontroller. The board is manufactured by STMicroelectronics.

On the board we connected Bluetooth HC-05 module, RFID reader and TFT Touch screen module. As a mobile device, for testing our Android project we used Samsung I9105 Galaxy S II Plus. All components of the hardware design are shown in Figure 3.1. In the next sections we will provide deeper explanations, about the components that we have used during the creation of our whole project.

3.1.1 STM32F4 Discovery board

The STM32F407 is based on the high-performance ARM® Cortex™-M4 32-bit RISC core operating at a frequency of up to 168 MHz. The Cortex-M4 core features a FPU (Cortex-M4F) single precision which supports all ARM single- precision data-processing instructions and data types [10]. It also implements a full set of

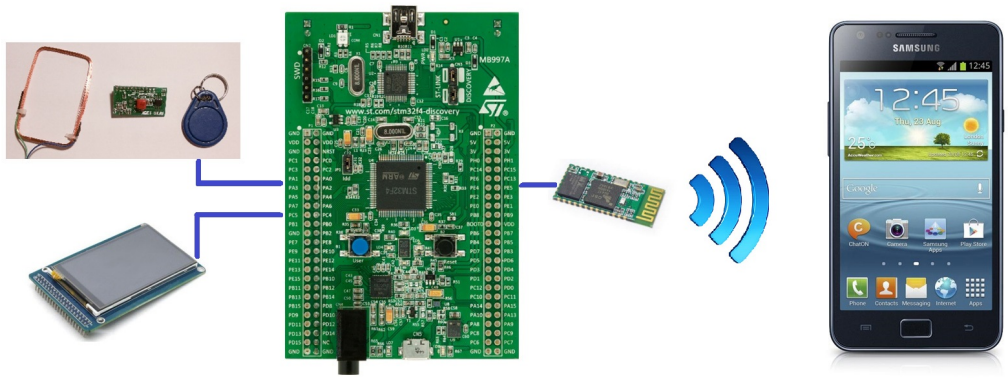


Figure 3.1: All components for the hardware design

DSP instructions and a memory protection unit (MPU) which enhances application security.

The STM32F407 incorporates high-speed embedded memories (Flash memory up to 1 Mbyte, up to 192Kbytes of SRAM), up to 4Kbytes of backup SRAM, and an extensive range of enhanced I/Os and peripherals connected to two APB buses, three AHB buses and a 32-bit multi-AHB bus matrix.

All devices offer three 12-bit ADCs, two DACs, a low-power RTC, twelve general-purpose 16-bit timers including two PWM timers for motor control, two general-purpose 32-bit timers, a true random number generator (RNG). They also feature standard and advanced communication interfaces.

- Up to three I2Cs
- Three SPIs, two I2Ss full duplex. To achieve audio class accuracy, the I2S peripherals can be clocked via a dedicated internal audio PLL or via an external clock to allow synchronization.
- Four USARTs plus two UARTs
- An USB OTG full-speed and a USB OTG high-speed with full-speed capability (with the ULPI),
- Two CANs
- An SDIO/MMC interface

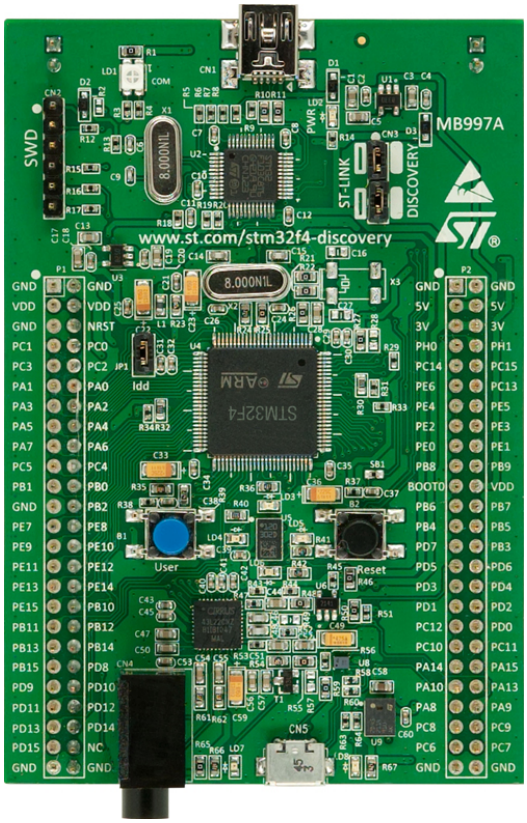


Figure 3.2: STM32F4 Discovery board

- Ethernet and the camera interface available on STM32F407VG devices only.

New advanced peripherals include an SDIO, an enhanced flexible static memory control (FSMC) interface (for devices offered in packages of 100 pins and more), and a camera interface for CMOS sensors.

The STM32F407VG family operates in the –40 to +105 °C temperature range from a 1.8 to 3.6 V power supply. The supply voltage can drop to 1.7 V when the device operates in the 0 to 70 °C temperature range using an external power supply supervisor. A comprehensive set of power-saving mode allows the design of low-power applications.

The STM32F407VG offers devices in various packages ranging from 64 pins to 176 pins. The set of included peripherals changes with the device chosen.

These features make the STM32F407VG microcontroller family suitable for a wide range of applications:

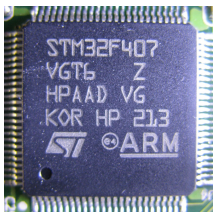


Figure 3.3: STM32F407VGT6

- Motor drive and application control
- Medical equipment
- Industrial applications: PLC, inverters, circuit breakers
- Printers, and scanners
- Alarm systems, video intercom, and HVAC
- Home audio appliances

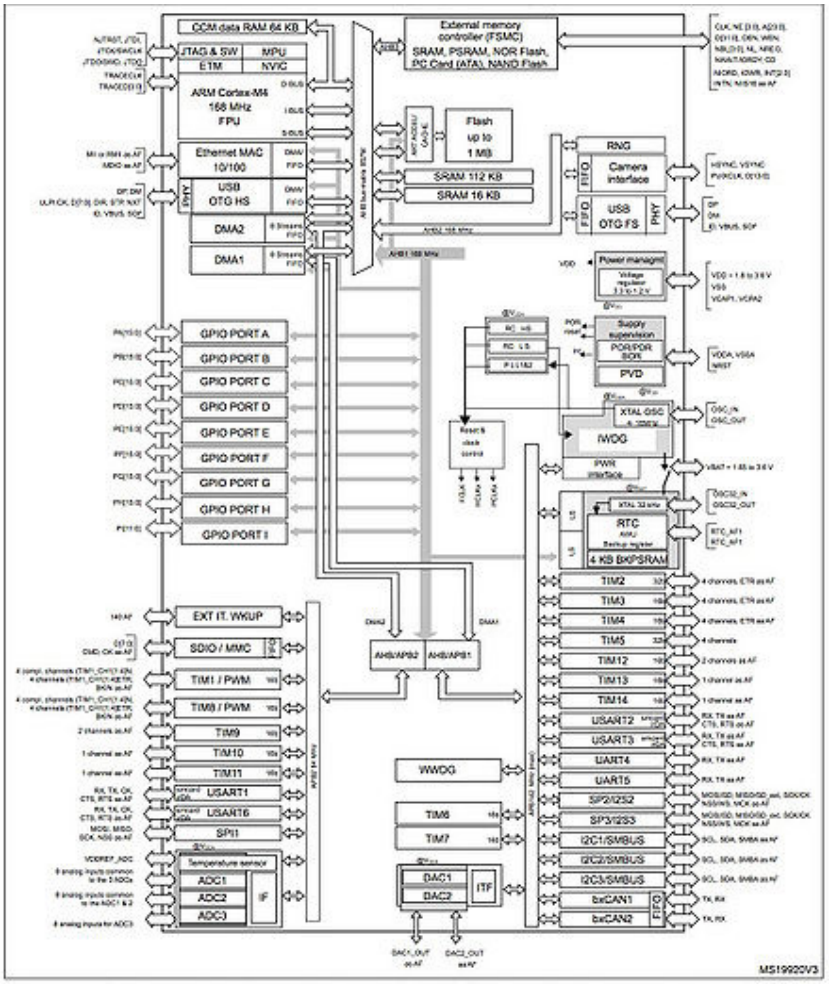
Figure 3.4 shows the STM32F407 block diagram.

ARM® Cortex™-M4F core with embedded Flash and SRAM

ARM® Cortex™-M4F core with embedded Flash and SRAM The ARM Cortex-M4F processor is the latest generation of ARM processors for embedded systems. It was developed to provide a low-cost platform that meets the needs of MCU implementation, with a reduced pin count and low-power consumption, while delivering outstanding computational performance and an advanced response to interrupts.

The ARM Cortex-M4F 32-bit RISC processor features exceptional code-efficiency, delivering the high-performance expected from an ARM core in the memory size usually associated with 8- and 16-bit devices.

The processor supports a set of DSP instructions which allow efficient signal processing and complex algorithm execution. Its single precision FPU (floating point unit) speeds up software development by using metalanguage development tools, while avoiding saturation [11]. The STM32F407VG family is compatible with all ARM tools and software.



1. The timers connected to APB2 are clocked from TIMxCLK up to 168 MHz, while the timers connected to APB1 are clocked from TIMxCLK either up to 84 MHz or 168 MHz, depending on TIMPRE bit configuration in the RCC_DCKCFGR register.
2. The camera interface and ethernet are available only on STM32F407xx devices.

Figure 3.4: STM32F407 Block Diagram

General-purpose I/Os

Each general-purpose I/O port has four 32-bit configuration registers (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR and GPIOx_PUPDR), two 32-bit data registers (GPIOx_IDR and GPIOx_ODR), a 32-bit set/reset register (GPIOx_BSRR), a 32-bit locking register (GPIOx_LCKR) and two 32-bit alternate function selection register (GPIOx_AFRH and GPIOx_AFRL).

The main features of GPIO are:

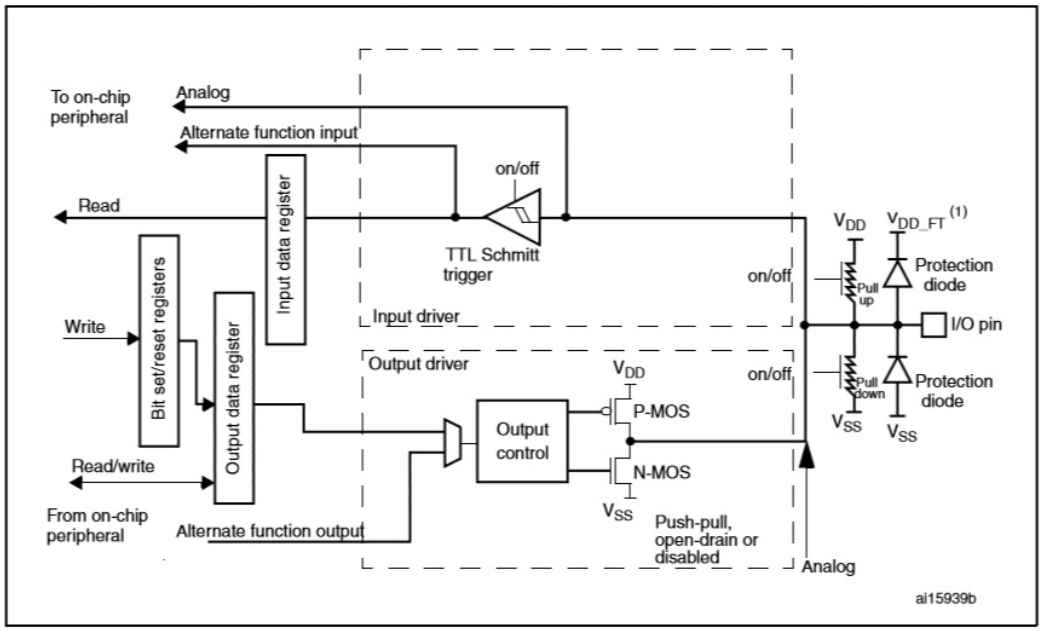
- Up to 16 I/Os under control
- Output states: push-pull or open drain + pull-up/down Output data from output data register (GPIOx_ODR) or peripheral (alternate function output)
- Speed selection for each I/O
- Input states: floating, pull-up/down, analog
- Input data to input data register (GPIOx_IDR) or peripheral (alternate function input)
- Bit set and reset register (GPIOx_BSRR) for bitwise write access to GPIOx_ODR
- Locking mechanism (GPIOx_LCKR) provided to freeze the I/O configuration
- Analog function
- Alternate function input/output selection registers (at most 16 AFs per I/O)
- Fast toggle capable of changing every two clock cycles
- Highly flexible pin multiplexing allows the use of I/O pins as GPIOs or as one of several peripheral functions

Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:

- Input floating
- Input pull-up
- Input-pull-down
- Analog
- Output open-drain with pull-up or pull-down capability
- Output push-pull with pull-up or pull-down capability

- Alternate function push-pull with pull-up or pull-down capability
- Alternate function open-drain with pull-up or pull-down capability

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words, half-words or bytes [11]. The purpose of the GPIOx_BSRR register is to allow atomic read/modify accesses to any of the GPIO registers. In this way, there is no risk of an IRQ occurring between the read and the modify access. Figure 3.5 shows the basic structure of a 5 V tolerant I/O port bit.



1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD}.

Figure 3.5: Basic structure of a five-volt tolerant I/O port bit

Nested vectored interrupt controller

The nested vector interrupt controller NVIC includes the following features:

- 82 maskable interrupt channels
- 16 programmable priority levels (4 bits of interrupt priority are used)

- low-latency exception and interrupt handling
- power management control
- implementation of system control registers

The NVIC and the processor core interface are closely coupled, which enables low latency interrupt processing and efficient processing of late arriving interrupts. All interrupts including the core exceptions are managed by the NVIC [11].

Universal synchronous asynchronous receiver transmitter

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator [11].

It supports synchronous one-way communication and half-duplex single wire communication. It also supports the LIN (local interconnection network), Smartcard Protocol and IrDA (infrared data association) SIR ENDEC specifications, and modem operations (CTS/RTS). It allows multiprocessor communication. High speed data communication is possible by using the DMA for multibuffer configuration.

Any USART bidirectional communication requires a minimum of two pins: Receive Data In (RX) and Transmit Data Out (TX).

RX: Receive Data Input is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

TX: Transmit Data Output. When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In single-wire and smartcard modes, this I/O is used to transmit and receive the data (at USART level, data are then received on SW_RX).

Through these pins, serial data is transmitted and received in normal USART mode as frames comprising:

- An Idle Line prior to transmission or reception
- A start bit

- A data word (8 or 9 bits) least significant bit first
- 0.5,1, 1.5, 2 Stop bits indicating that the frame is complete
- This interface uses a fractional baud rate generator - with a 12-bit mantissa and 4-bit fraction
- A status register (USART_SR)
- Data Register (USART_DR)
- A baud rate register (USART_BRR) - 12-bit mantissa and 4-bit fraction.
- A Guardtime Register (USART_GTPR) in case of Smartcard mode.

Figure 3.6 shows the USART block diagram.

AHB/APB bridges

The two AHB/APB bridges, APB1 and APB2, provide full synchronous connections between the AHB and the two APB buses, allowing flexible selection of the peripheral frequency [11]. After each device reset, all peripheral clocks are disabled (except for the SRAM and Flash memory interface). Before using a peripheral you have to enable its clock in the RCC_AHBxENR or RCC_APBxENR register.

3.1.2 HC-05 Bluetooth module

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup.

Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH(Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mm x 27mm [12].

Hardware features

- Typical -80dBm sensitivity

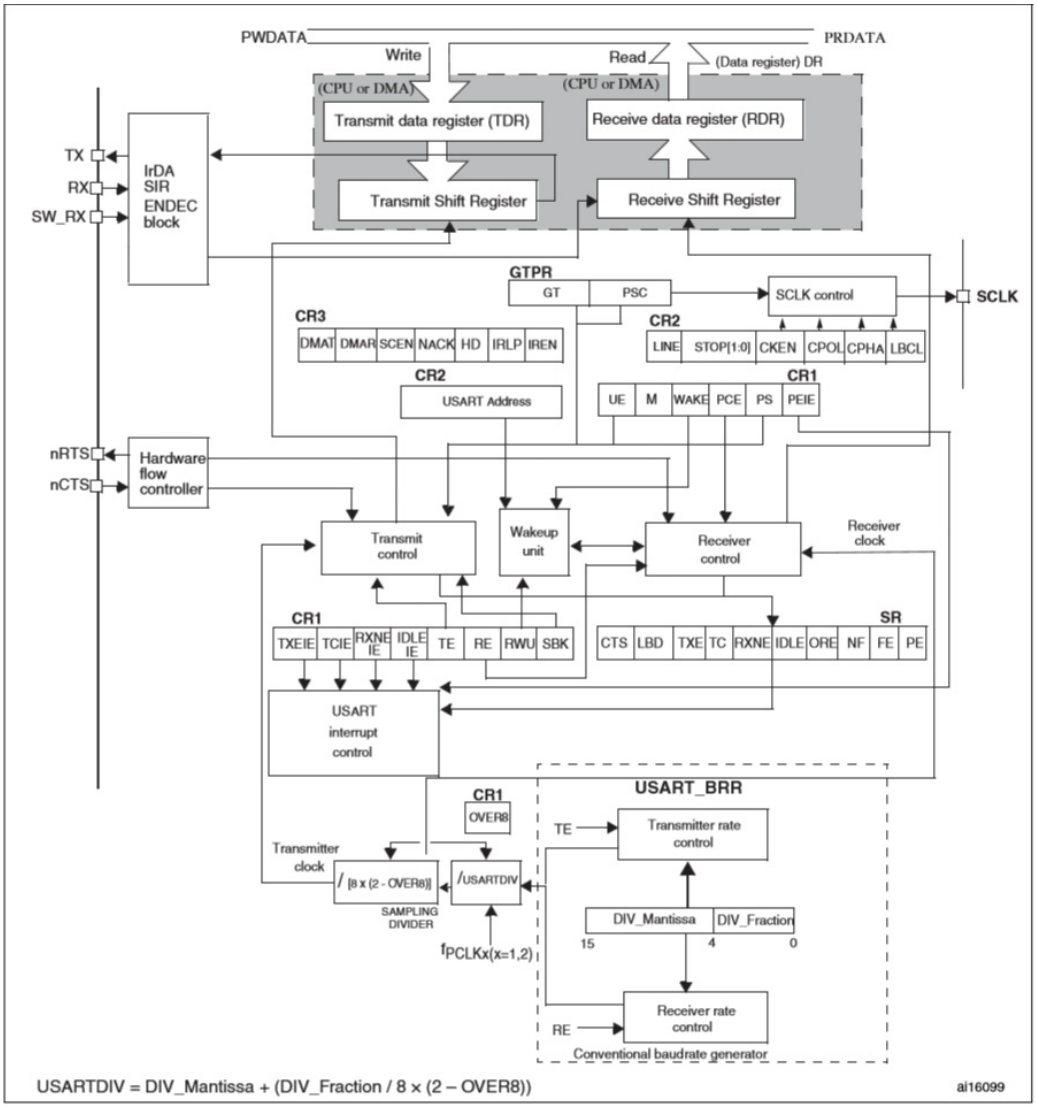


Figure 3.6: USART block diagram

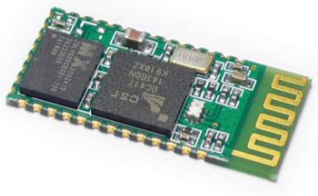


Figure 3.7: Bluetooth HC-05

- Up to +4dBm RF transmit power
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- PIO control
- UART interface with programmable baud rate
- With integrated antenna
- With edge connector

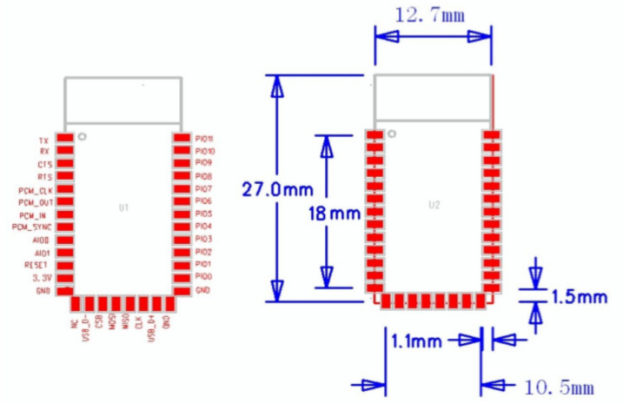


Figure 3.8: Bluetooth Pins and Dimensions

Software features

- Default Baud rate: 38400, Data bits:8, Stop bit:1, Parity:No parity, Data control: has. Supported baud rate: 9600, 19200, 38400, 57600, 115200, 230400, 460800.
- Given a rising pulse in PIO0, device will be disconnected.
- Status instruction port PIO1: low-disconnected, high-connected;
- PIO10 and PIO11 can be connected to red and blue led separately. When master and slave are paired, red and blue led blinks 1time/2s in interval, while disconnected only blue led blinks 2times/s.

- Auto-connect to the last device on power as default.
- Permit pairing device to connect as default.
- Auto-pairing PINCODE:"0000" as default
- Auto-reconnect in 30 min when disconnected as a result of beyond the range of connection.

Figure 3.9 shows the HC-05 block diagram.

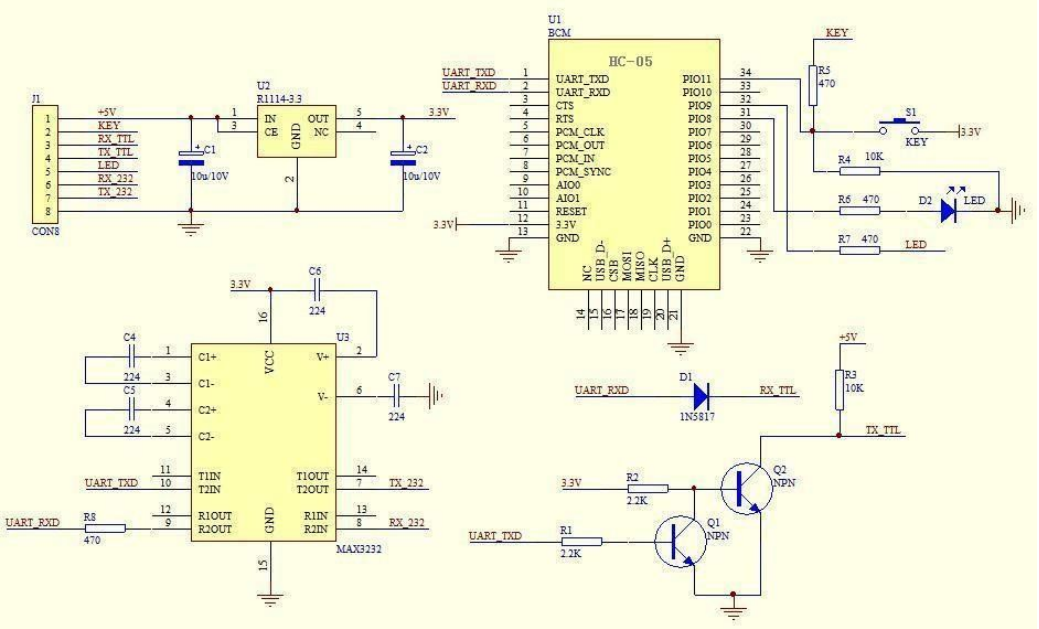


Figure 3.9: HC-05 block diagram

3.1.3 RFID module

For reading the RFID tags we used 125KHz mini-module. With an extremely basic information for the functionality of the module that were in the product description, additional information about the module were not found. It is compatible with ID cards of 64 bit read-only EM4100, TK4100, and EM4102 series. This module outputs original card information (Manchester code) and ascii codes which can be directly displayed on the computer.

- Voltage: 3 - 5V DC
- Current: <50mA
- Output format: Baud rate 9600, Data bits 8, stop bit 1
- Manchester code output:
- Reading Distance: key card> 5cm, thin card> 9cm, thick card> 11cm, with no blind spots
- Operation temperature: 0 70 degrees Celsius
- Size: 38mm * 20mm

On Figure 3.10 are shown the RFID antenna, RFID module and RFID tag.

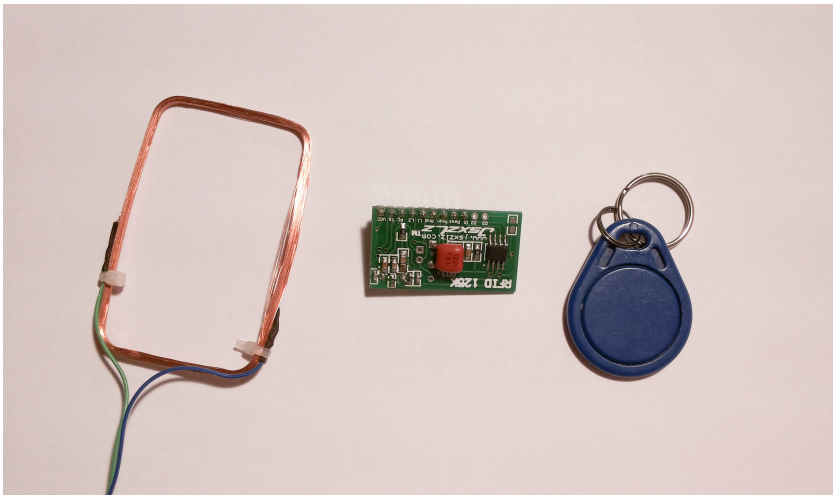


Figure 3.10: RFID Antenna, RFID Module and RFID Tag

3.1.4 TFT touch screen

For displaying information and perception of touch, we used the TFT module. The manufacturer is unknown. In the module are soldered LCD screen size of 3.2 inches, the display controller SSD1289 LCD screen, touch screen controller ADS7843 and slot for SD memory cards. On Figure 3.11 is presented our TFT touch screen.

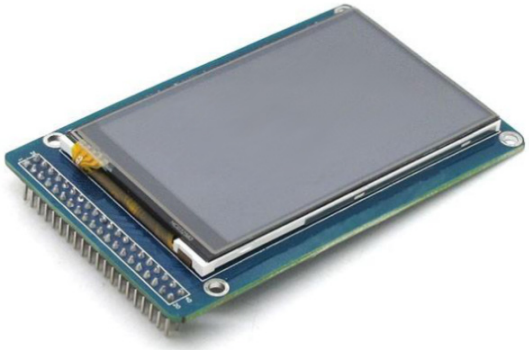


Figure 3.11: TFT Touch Screen

3.1.5 Mobile device

For testing our Android application we have used Samsung I9105 Galaxy S II Plus with the next features [13]:

- OS: Android OS, v4.1.2 (Jelly Bean), upgradable to v4.2.2 (Jelly Bean)
- Chipset: Broadcom BC28155
- CPU: Dual-core 1.2 GHz
- GPU: Broadcom VideoCore IV
- Java: Yes, via Java MIDP emulator

Figure 3.12: Samsung I9105 Galaxy S II Plus

3.2 Software solution

The software solution is consisted of two parts. The first part is a CoIDE project and the second part is Android application.

Our final goal was to build a system in which the Bluetooth module will send information to the mobile device,and the information that the mobile device receives, to be somehow displayed in an Android application. In addition, via Bluetooth we are not sending a previously defined string that does not change. Via Bluetooth we are sending an information only when it comes to a communication between the RFID tag and RFID antenna. If the RFID tag exists in the *Table of valid RFID tags* that is a list in which are defined all the tags the professor has, then the information will be shown in the Android application.

The first part is a CoIDE project, the embedded system for access control, which we have programmed on the STM32F4 Discovery Board. For programming this project, we have used the C programming language. The project was developed on the basis of an existing code, which my mentor had written in the past.

For programming the second part, the Android application, we used some parts of the code from the source [14].

3.2.1 CooIDE Application

In our configuration, we have connected the Bluetooth module on UART1 using the STM32F4's ports PB6(TX) and PB7(RX). The HC-05 bluetooth module is used as master in the communication and the android phone will be the slave. The RFID module is using STM32F4's port PB11. The wiring is presented on Figure 3.13.

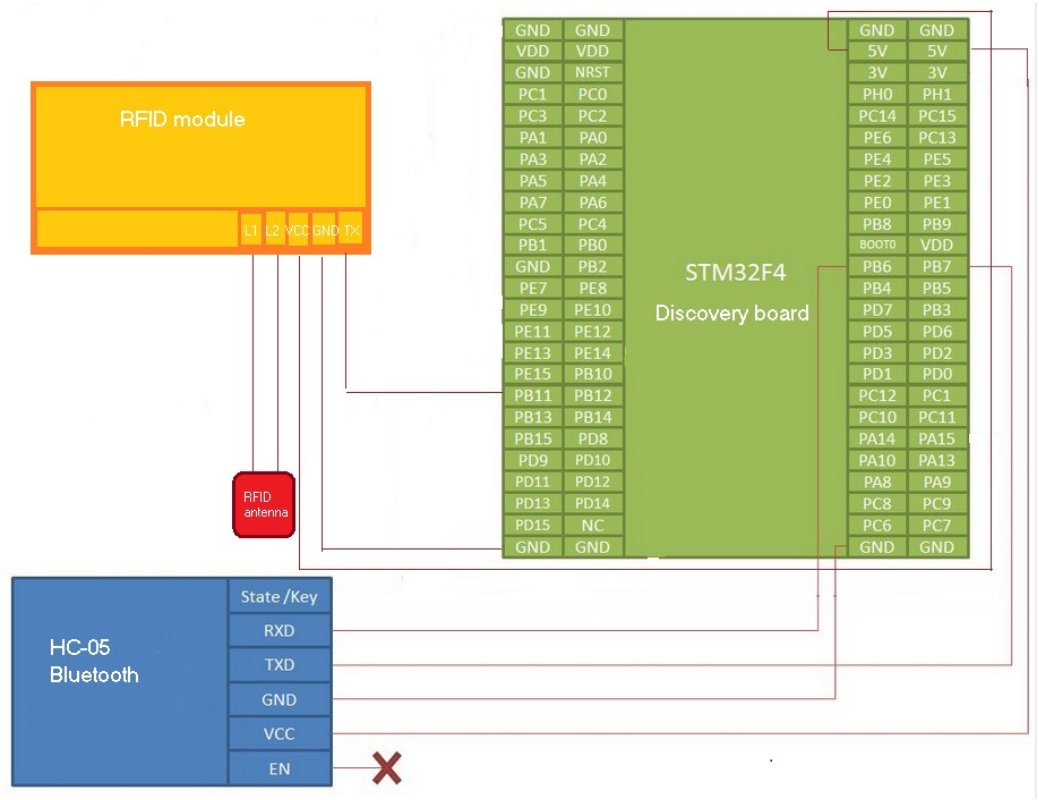


Figure 3.13: CoIDE application wiring

FreeRTOS overview

Microcontrollers (MCUs) that contain an ARM Cortex-M3 core are ideally suited to deeply embedded real-time applications. Typically, applications of this type include a mix of both hard and soft real-time requirements [15].

Soft real-time requirements are those that state a time deadline – but breaching the deadline would not render the system useless. For example, responding to

keystrokes too slowly may make a system seem annoyingly unresponsive without actually making it unusable.

Hard real-time requirements are those that state a time deadline – and breaching the deadline would result in absolute failure of the system. For example, a driver's airbag would be useless if it responded to crash sensor inputs too slowly.

FreeRTOS is a real-time kernel (or real-time scheduler) on top pf which Cortex-M3 microcontroller applications can be built to meet their hard real-time requirements. It allows Cortec-M3 microcontroller applications to be organized as a collection of independent threads of execution. As most Cortex-M3 microcontroller have only one core, in reality only a single thread can be executing at any one time. The kernel decides which thread should be executing by examining the priority assigned to each thread by the application designer. In the simplest case, the application designer could assign higher priorities to threads that implement soft real-time requirements. This would ensure that hard real-time threads are always executed ahead of soft real-time threads, but priority assignment decisions are not always that simplistic.

The Cortex-M3 port includes all the standard FreeRTOS features:

- Pre-emptive or co-operative operation
- Very flexible task priority assignment
- Queues
- Binary semaphores
- Counting semaphores
- Recursive semaphores
- Mutexes
- Tick hook functions
- Idle hook functions
- Stack overflow checking
- Trace took macros

- Optional commercial licensing and support

FreeRTOS also manages interrupt nesting, and allows interrupts above a user-definable priority level to remain unaffected by the activity of the kernel. Using FreeRTOS will not introduce any additional timing jitter or latency for these interrupts.

Creating Access Control project in CoIDE

As we explained before in section 2.4.4 Creating project in CoIDE, for creating a new project based on the STM32F4 Discovery Board, more specifically STM32F407VG Discovery Board, we selected "Create a New Project" from the Welcome page, and from the wizard pop up window instead of selecting "Board" we selected "Chip". In the next step as manufacturer we selected "ST", Series – "STM32F4x" and Device – "STM32F407VG". As the final step, we clicked "Finish" button, and with that final step we had an empty project, only with main.c file.

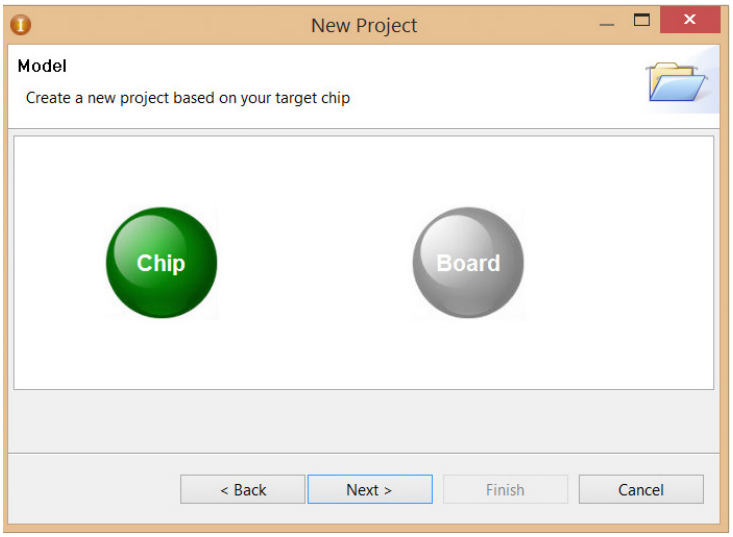


Figure 3.14: Creating Access Control Project

For programming my Access Control project we have used the next Basic Components from Repository, shown in Figure 3.16:

1. Common components:

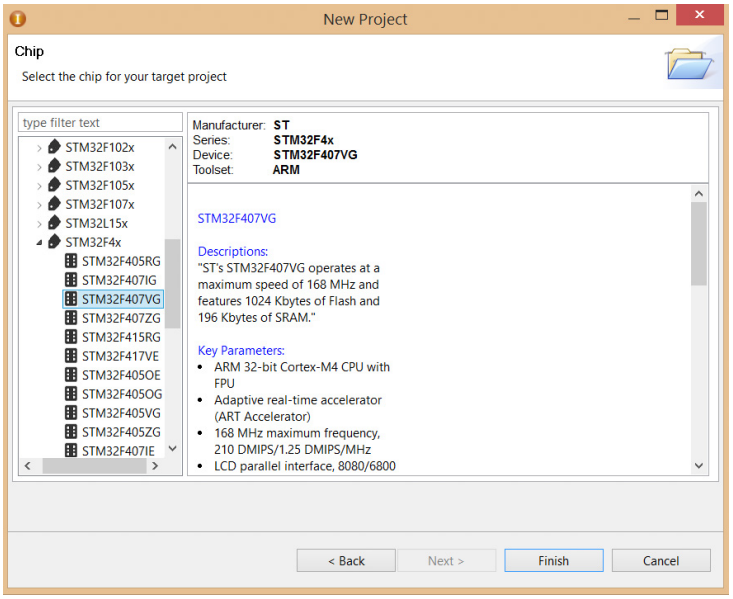


Figure 3.15: Creating Access Control Project

- M4 CMSIS Core (CMSIS core for Cortex M4 V 3.01)

2. Boot components:

   - CMSIS BOOT (CMSIS BOOT for STM32F4x series)

3. Peripheral components:

   - RCC (Reset and clock control for STM32F4xx)
   - GPIO (General Purpose Input/Output for STM32F4xx)
   - EXTI (External Interrupt/Line Controller for STM32F4xx)
   - SPI (Serial peripheral interface for STM32F4xx)
   - I2C (Inter-integrated circuit interface for STMF4xx)
   - DMA (Direct Memory Access for STM32F4xx)
   - DAC (General Purpose Input/Output for STM32F4xx)
   - MISC (Nested vectored interrupt controller for STM32F4xx)
   - SYSCFG (System configuration controller for STM32F4xx)

4. RTOS components:
    - FreeRTOS
5. PERIPHERAL.ST_FW components:
    - STM32F4-Discovery

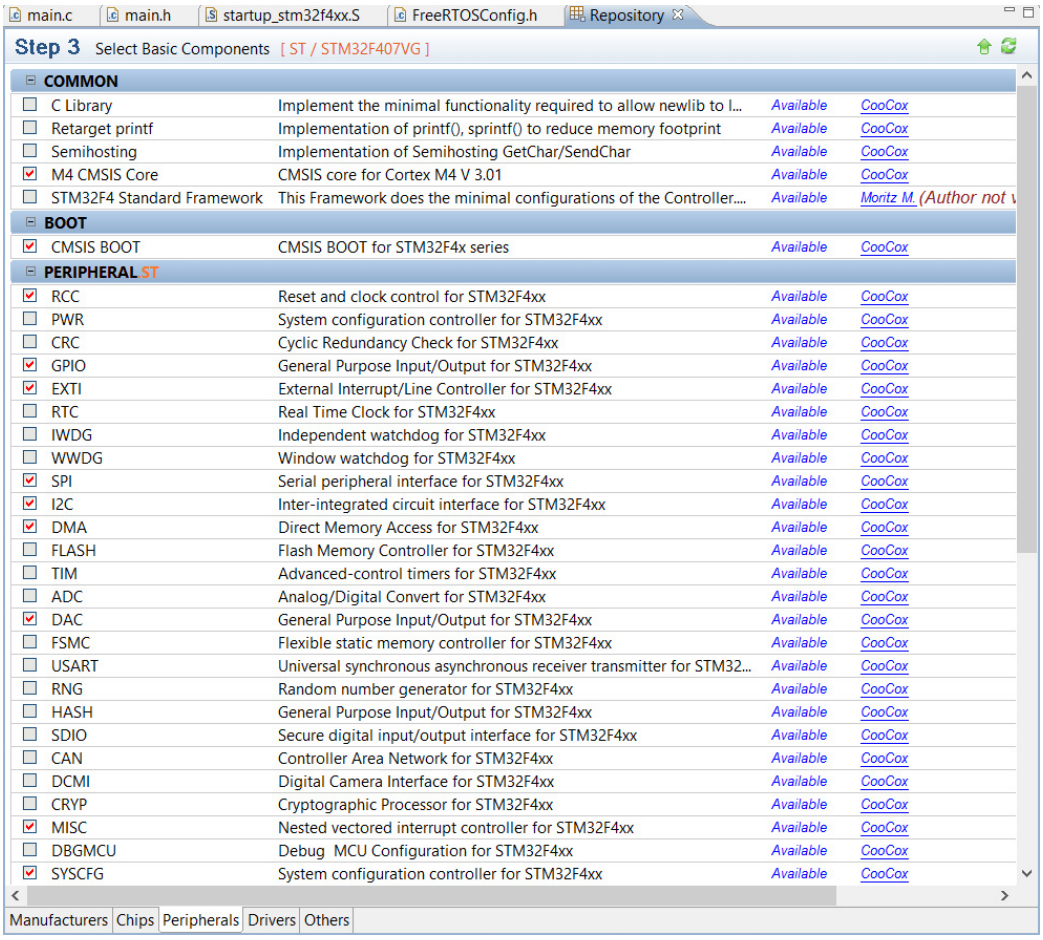


Figure 3.16: Repository

How CoIDE project works

First, when we start our CoIDE project on the TFT Touch screen, we can see a few messages that inform us for the initialization of the peripherals, RFID, Touch

panel, Starting FreeRTOS, Creating queues and Creating tasks. This is shown in Figure 3.17.

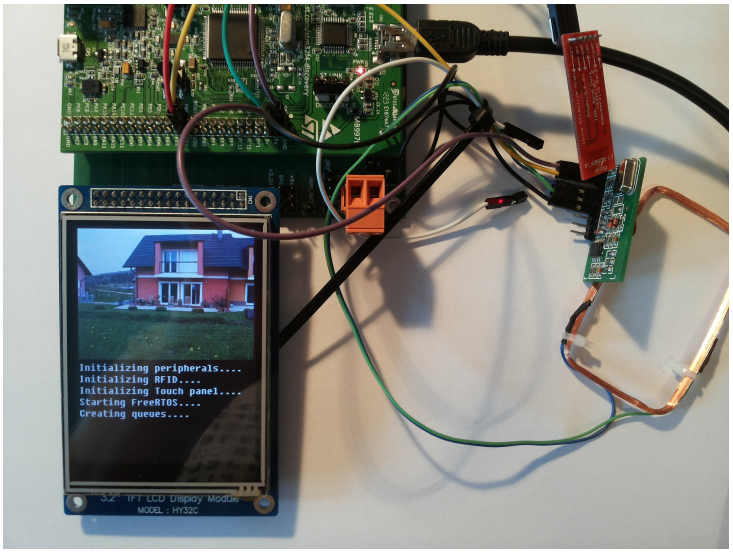


Figure 3.17: Initialization

Second, when the initialization is finished, on Figure 3.18, we can see how our project looks like.

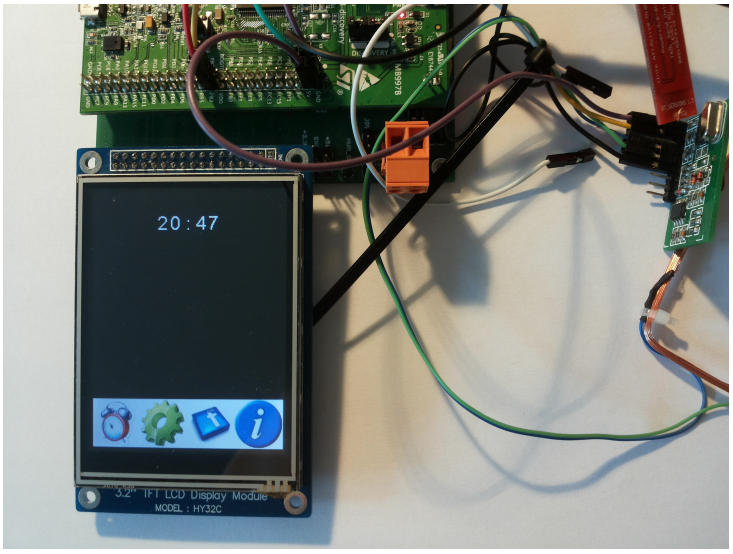


Figure 3.18: CoIDE project

We can see the time and if the RFID tag, comes near the RFID antenna, the string will be shown, Figure 3.19.

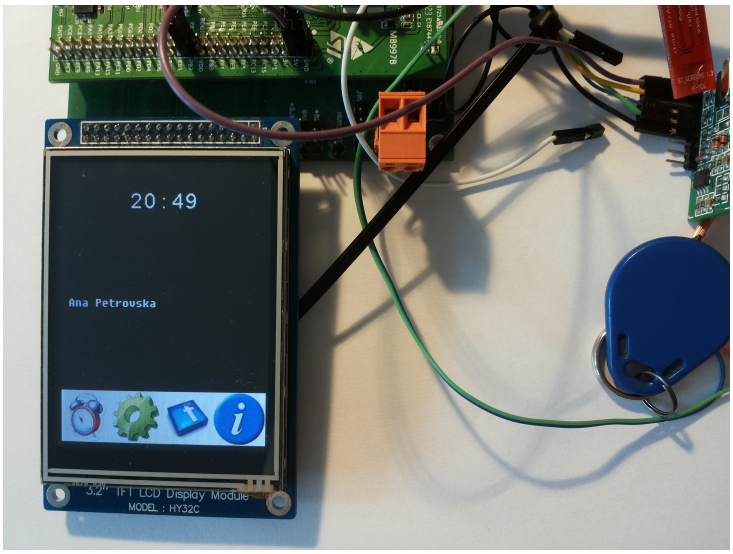


Figure 3.19: CoIDE project showing the string

With clicking on the Clock Picture we can change the time. We have a choice to change the hours and also the minutes. When we have finished, we click on the green button to save the change, or with clicking on the red button, we return in the previous state, without changed clock, Figure 3.20.

Sending string over UART

In this chapter we will explain the code for sending a string over UART.

```
void init_USART1(uint32_t baudrate)
```

The function *init_ USART1* initializes the USART1 peripheral. The argument *uint32_ t baudrate* is the baudrate at which the USART is supposed to operate.

This is a concept is working with the libraries provided by ST to make development easier. They are developing the projects with something similar to classes, called TypeDefs. With TypeDefs we actually just define the common parameters that every peripheral needs to work correctly. The typedef keyword allows the programmer to create new names for types such as, for example, int. They make

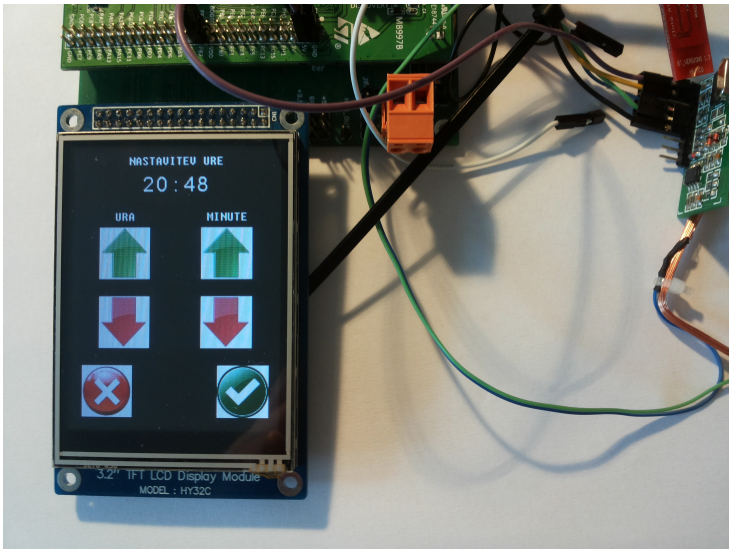


Figure 3.20: Changing the clock

our programming easier. The most important purpose of having types is to make sure that variables are always used in the way that they were intended to be used.

```
GPIO_InitTypeDef GPIO_InitStruct;
USART_InitStruct;
NVIC_InitTypeDef NVIC_InitStructure;
```

GPIO_InitTypeDef GPIO_InitStruct is a TypeDef for the GPIO pins used as TX and RX. *USART_InitTypeDef USART_InitStruct* this is for the USART1 initilization. *NVIC_InitTypeDef NVIC_InitStructure* this is used to configure the NVIC.

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
```

This line enables APB2 peripheral clock for USART1. Only USART1 and USART6 are connected to APB2, the other USARTs are connected to APB1.

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
```

This line of code enables the peripheral clock for the pins used by USART1. Pin PB6 will be used for TX and pin PB7 for RX.

```
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOB, &GPIO_InitStruct);
```

This sequence sets up the TX and RX pins so they work correctly with the USART1 peripheral. With the first line *GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7* we initialize pins 6 (TX) and 7 (RX), which will be used in our project . With the second line *GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF* we are configuring the pins as alternate function, so the USART peripheral can have access to them. The third line, *GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50 MHz* defines the IO speed. This has no connection with the baudrate. *GPIO_InitStruct. GPIO_OType = GPIO_OType_PP* defines the output type as push pull mode. *GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP* this activates the pullup resistors on the IO pins. With the last line, *GPIO_Init(GPIOB, &GPIO_InitStruct)*, all the values are passed to the GPIO_Init() function which sets the GPIO registers.

```
GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_USART1);
GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_USART1);
```

The RX and TX pins are now connected to their AF so that the USART1 can take over control of the pins.

```
USART_InitStruct.USART_BaudRate = baudrate;
USART_InitStruct.USART_WordLength = USART_WordLength_8b;
USART_InitStruct.USART_StopBits = USART_StopBits_1;
USART_InitStruct.USART_Parity = USART_Parity_No;
USART_InitStruct.USART_HardwareFlowControl = USART_
HardwareFlowControl_None;
```

```
USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
USART_Init(USART1, &USART_InitStruct);
```

With this code the USART_InitStruct is used to define the properties of USART1. With the first line, *USART_InitStruct.USART_BaudRate = baudrate* the baudrate is set to the value we passed into this init function. We want the data frame size to be 8 bits. With the second code line *USART_InitStruct.USART_WordLength = USART_WordLength_8b* we are defining the data frame size. With the next line, *USART_InitStruct.USART_StopBits = USART_StopBits_1* we are defining 1 stop bit. We do not want a parity bit, *USART_InitStruct.USART_Parity = USART_Parity_No*, we do not want flow control, *USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None* and we want to enable the transmitter and the receiver *USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx*. With the last line, *USART_Init(USART1, &USART_InitStruct)*, all the properties are passed to the USART_Init function which takes care of all the bit settings.

```
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

With this code the USART1 receive interrupt is enabled and the interrupt controller is configured to jump to the USART1_IRQHandler() function if the USART1 receive interrupt occurs. With the first line *USART_ITConfig(USART1, USART_IT_RXNE, ENABLE)* we are enabling the USART1 receive interrupt. With the second line *NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn* we are configuring the USART1 interrupts. The third line, *NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0* sets the priority group of the USART1 interrupts and the next line, *NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0* sets the subpriority inside the group. With the next line, *NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE* the USART1 interrupts are globally enabled. With

the last line *NVIC_Init(&NVIC_InitStructure)* the properties are passed to the NVIC_Init function.

```
USART_Cmd(USART1, ENABLE);
```

Finally, this line enables the complete USART1 peripheral.

All this lines of code, are inside *init_USART1* function, the funcion initializes the USART1 peripheral.

The next function that I will describe is the interruption handler.

```
void USART1_IRQHandler(void){
  if( USART_GetITStatus(USART1, USART_IT_RXNE) ){
    static uint8_t cnt = 0;
    char t = USART1->DR;
    if( (t != 'n') && (cnt < MAX_STRLEN) ){
      received_string[cnt] = t;
      cnt++;
    }
    else{ // otherwise reset the character counter
      cnt = 0;
    }
  }
}
```

This is the interrupt request handler (IRQ) for all USART1 interrupts. With the if statement *if(USART_GetITStatus(USART1, USART_IT_RXNE))* we are checking if the USART1 receive interrupt flag was set. We are using *static uint8_t cnt* counter to determine the string length. With the next line, *char t = USART1->DR*, the character from the USART1 data register is saved in t. With the next if/else statement, we are checking if the received character is not the LF character (used to determine end of string) or the if the maximum string length has been been reached, *(t != 'n') && (cnt <MAX_STRLEN)*, otherwise reset the character counter, *cnt = 0*.

```
void UARTSend(const unsigned char *pucBuffer, unsigned long
 ulCount)
{
  while(ulCount--)
  {
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) ==
     RESET)
    {
    }
    USART_SendData(USART1, (uint8_t) *pucBuffer++);
  }
}
```

Finally, we came to the function *UARTSend(const unsigned char *pucBuffer, unsigned long ulCount)*. That is the function that sends a string to the UART. The first parametar *pcBuffer is a buffer, temporary storage area. The purpose of most buffers is to act as a holding area, enabling the CPU to manipulate data before transferring it to a device. The second parametar ulCount is the buffer's length.

The function *UARTSend(const unsigned char *pucBuffer, unsigned long ulCount)* is written inside the task *vLCDTask*. The part of the code, is shown below.

```
void vLCDTask( void *pvParameters )
{
  xLCDMessage xMessage;
  LCD_Clear(BLACK);
  xLCDScreenState = LCD_NORMAL;
  for( ;; )
  {
    /* Wait for a message to arrive that requires displaying.*/
    while( xQueueReceive( xLCDQueue, &xMessage,
    portMAX_DELAY ) != pdPASS );
      switch( xLCDScreenState ) {
        case LCD_NORMAL:
```

```
            ScreenNormal(&xMessage);
            break;
          ......
        }
    }
}
```

```
void ScreenNormal (xLCDMessage *xpMessage) {
  switch (xpMessage->xMsgType) {
    ....
    case UNAME:
            strcpy(str_output, xpMessage->pcMessage);
            strcat(str_output, str_two);
      init_USART1(BT_BAUD);
      LCD_StringLine(8,150,xpMessage->pcMessage);
      UARTSend(str_output, sizeof(str_output));
      Delay(1000000);
      break;
    .....
  }
}
```

3.2.2 Android Aplication

ADT Bundle overview

For developing our Android application we used ADT Bundle. It includes the essential Android SDK components and a version of the Eclipse IDE with built-in ADT to streamline our Android application development [16]. The Android SDK provides us with the API libraries and developer tools necessary to build, test, and debug applications for Android. The Eclipse ADT bundle includes everything we have needed to begin developing our application:

- Eclipse + ADT plugin

- Android SDK Tools
- Android Platform-tools
- A version of the Android platform
- A version of the Android system image for the emulator

How Android application works

Our Android application is presented on Figure 3.21. As we can see , when we start the application, we have three active buttons *Quit*, *Turn Bt On* and *About*.

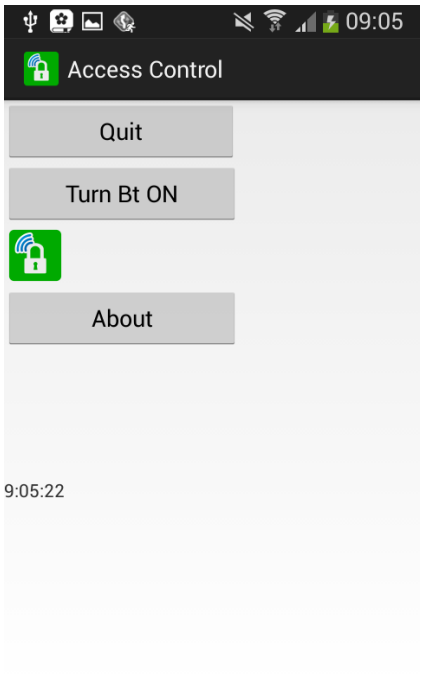


Figure 3.21: Android application

If we click on the first button, we will quit the application. If we click on the button *Turn Bt ON*, a pop up window will open. That pop up window is shown in 3.22. The pop up window is asking for permission to turn on the Bluetooth on the mobile device. We have two options. The first option is to click on *Yes* button, and with that we will have turned on the Bluetooth. The second option, is to click

No. If we decide not to turn on the Bluetooth, we will be returned back on the previous state, Figure 3.21.

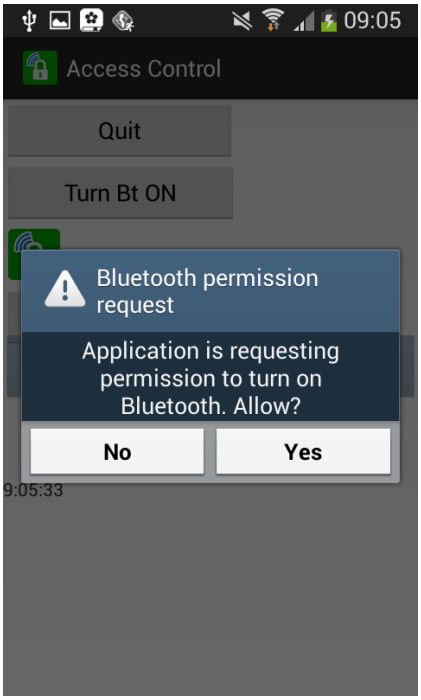


Figure 3.22: Android application

If we decide to turn on the Bluetooth, our Android application will be in a state that is presented in Figure 3.23. In this state, the button *Turn Bt On* is replaced with the new button, *Search Bt Devices*.

In this example, if we click this button, a new pop up window will be shown, Figure 3.23. Here, we receive a list of all, in that moment, active Bluetooth devices. With selecting the preferable device, the Bluetooth of our mobile device, is paired with the selected device. In this case we want to pair with our Bluetooth module *HC-05*. When we had paired the Bluetooth of our mobile device with the *HC-05* module, for the first time, we were asked to write a pin code. If we enter the correct pin code, the Bluetooth modules are finally paired.

After pairing our Bluetooth modules, we are on the next state of our application, shown in Figure 3.25. In this state, we can see that we have a choice to disconnect from the connected Bluetooth device, our Bluetooth module, *HC-05*.

In this state, if the RFID tag, comes near the RFID antenna, the string will be

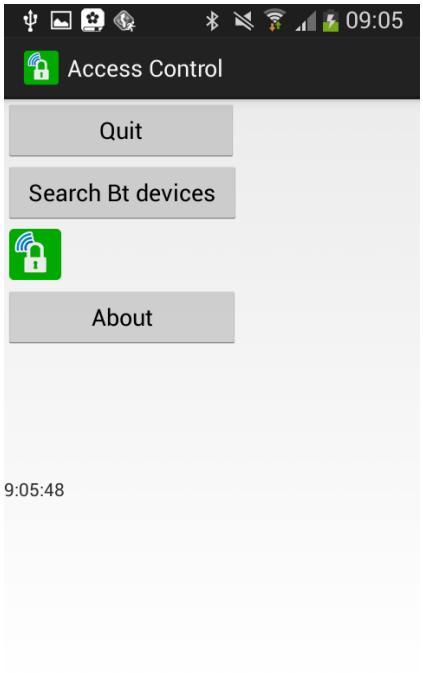


Figure 3.23: Android application

shown, Figure 3.26. If we press the disconnect button, we will be returned in the state of our application, which is shown in Figure 3.23.

If we press the button *About* a pop up window, presented in Figure 3.27, will be shown.

Finally, when we want to quit the application, we press *Quit*, and with that a pop up window, Figure 3.28, is shown. Because, during the time when we are using the application, the Bluetooth of the mobile device is turned on, the last pop up window asks us if we want to disable the Bluetooth adapter.

Communication between HC-05 and mobile device

The code described in this section does the biggest part of the work for setting up and managing Bluetooth connections with other devices.

The Android platform includes support for the Bluetooth network stack, which allows a device to wirelessly exchange data with other Bluetooth devices. The application framework provides access to the Bluetooth functionality through the Android Bluetooth APIs. These APIs let applications wirelessly connect to other

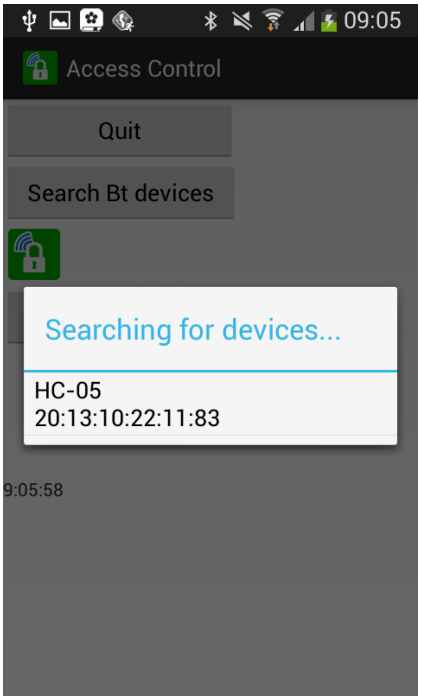


Figure 3.24: Android application

Bluetooth devices, enabling point-to-point and multipoint wireless features.

Using the Bluetooth APIs, an Android application can perform the following:

- Scan for other Bluetooth devices
- Query the local Bluetooth adapter for paired Bluetooth devices
- Establish RFCOMM channels
- Connect to other devices through service discovery
- Transfer data to and from other devices
- Manage multiple connections

All of the Bluetooth APIs are available in the android.bluetooth package. Here's a summary of the classes and interfaces that we had used for our Bluetooth connection:

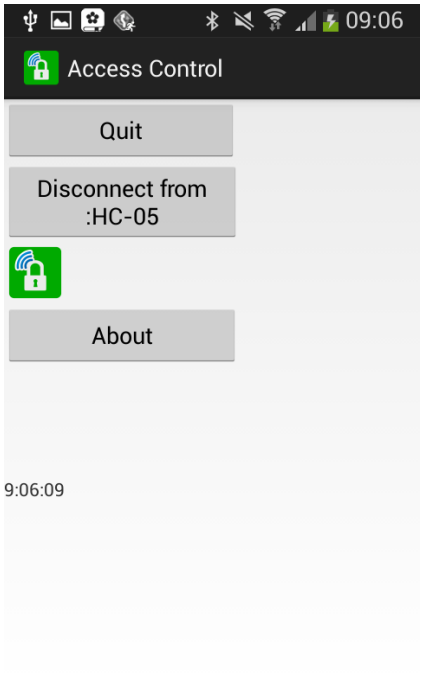


Figure 3.25: Android applications

```
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
```

BluetoothAdapter - Represents the local Bluetooth adapter (Bluetooth radio). The BluetoothAdapter is the entry-point for all Bluetooth interaction. Using this, we discovered other Bluetooth devices, query a list of bonded (paired) devices, instantiate a BluetoothDevice using a known MAC address, and create a BluetoothServerSocket to listen for communications from other devices.

BluetoothDevice - Represents a remote Bluetooth device. We used this to request a connection with a remote device through a BluetoothSocket or query information about the device such as its name, address, class, and bonding state.

BluetoothSocket - Represents the interface for a Bluetooth socket (similar to a TCP Socket). This is the connection point that allows an application to exchange data with another Bluetooth device via InputStream and OutputStream.

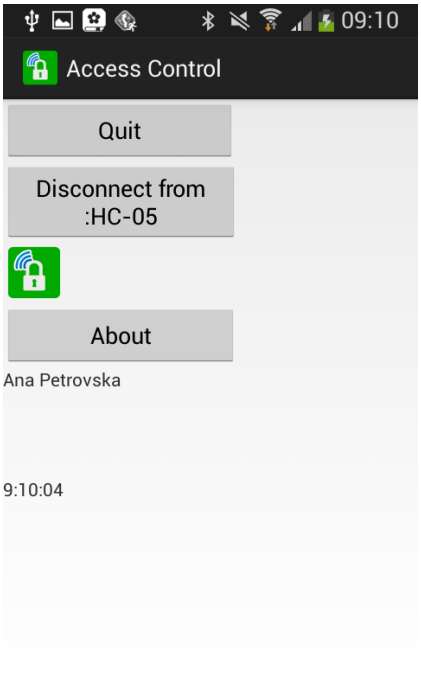


Figure 3.26: Android application

```
<manifest ... >
    <uses-permission android:name="android.permission.
    BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.
    BLUETOOTH" />
    <uses-permission android:name="android.permission.
    WAKE_LOCK" />
</manifest ... >
```

In order to use Bluetooth features in our application, we had to declare the Bluetooth permission BLUETOOTH. We needed this permission to perform any Bluetooth communication, such as requesting a connection, accepting a connection, and transferring data. We have declared the Bluetooth permissions in our application manifest file.

Before our application can communicate over Bluetooth, we have to to verify that Bluetooth is supported on the device, and if so, we had to ensure that it is

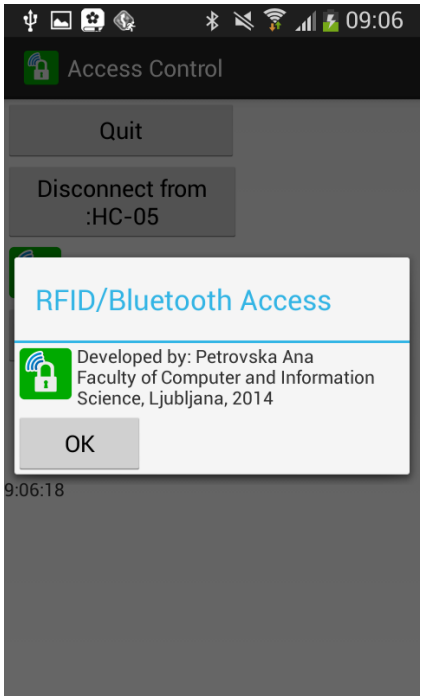


Figure 3.27: Android application

enabled.

If Bluetooth is not supported, then we have to gracefully disable any Bluetooth features. If Bluetooth is supported, but disabled, then we can request that the user enable Bluetooth without leaving your application. This setup is accomplished with using the BluetoothAdapter.

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.
getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device does not support Bluetooth
}
```

The BluetoothAdapter is required for any and all Bluetooth activity. To get the BluetoothAdapter, we call the static getDefaultAdapter() method. This returns a BluetoothAdapter that represents the device's own Bluetooth adapter (the Bluetooth radio).

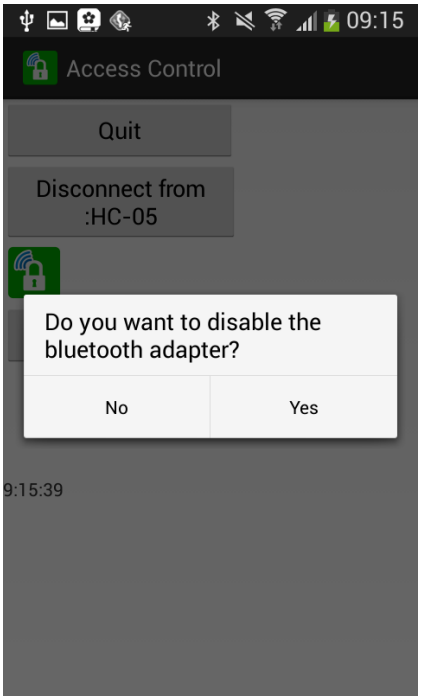


Figure 3.28: Android application

```
if (!mBluetoothAdapter.isEnabled())
```

Next, we have to ensure that Bluetooth is enabled. Call isEnabled() to check whether Bluetooth is currently enable. If this method returns false, then Bluetooth is disabled.

Using the BluetoothAdapter, we can find remote Bluetooth devices either through device discovery or by querying the list of paired (bonded) devices.

Device discovery is a scanning procedure that searches the local area for Bluetooth enabled devices and then requesting some information about each one (this is sometimes referred to as "discovering," "inquiring" or "scanning"). However, a Bluetooth device within the local area will respond to a discovery request only if it is currently enabled to be discoverable. If a device is discoverable, it will respond to the discovery request by sharing some information, such as the device name, class, and its unique MAC address. Using this information, the device performing discovery can then choose to initiate a connection to the discovered device.

Once a connection is made with a remote device for the first time, a pairing request is automatically presented to the user. When a device is paired, the basic information about that device (such as the device name, class, and MAC address) is saved and can be read using the Bluetooth APIs. Using the known MAC address for a remote device, a connection can be initiated with it at any time without performing discovery (assuming the device is within range).

```
// firstSearch for all devices already paired
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter
.getBondedDevices();
        //If there are paired devices
        if (pairedDevices.size() > 0) {
                // Loop through paired devices
           for (BluetoothDevice device : pairedDevices){
                        addDevice(device);
           }
        }
```

Before performing device discovery, its worth querying the set of paired devices to see if the desired device is already known. To do so, call getBondedDevices(). This will return a Set of BluetoothDevices representing paired devices.

```
if (mBluetoothAdapter.startDiscovery())
```

To start discovering devices, simply call startDiscovery(). The process is asynchronous and the method will immediately return with a boolean indicating whether discovery has successfully started. The discovery process usually involves an inquiry scan of about 12 seconds, followed by a page scan of each found device to retrieve its Bluetooth name.

Most applications have a server and a client side. A client usually establishes the Bluetooth connection to the master and requests the transfer of some kind of data. Thus, the master and the client sides of a Bluetooth service are different. It is noteworthy that the client/server principle of the Bluetooth profile should not be confused with the master/slave concept of the lower Bluetooth protocol layers.

The master/slave concept is used to control the piconet, that is, who is allowed to send and at the which time, while the client sever principle describes a service and the user of a service. As we said before, our HC-05 bluetooth module is used as master in the communication and the Android phone is the slave. Also Android application is programmed to be the client, because we only receive a string from the Bluetooth module, that means that we are the user of a service.

In order to initiate a connection with a remote device, and connect as a client, firstly we have to obtain a BluetoothDevice object that represents the remote device. After that we use the BluetoothDevice to acquire a BluetoothSocket and initiate the connection.

```
try {
  tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
} catch (IOException e) { }
```

First, we use the BluetoothDevice, get a BluetoothSocket by calling createRfcommSocketToServiceRecord(UUID).

This initializes a BluetoothSocket that will connect to the BluetoothDevice. The UUID passed here must match the UUID used by the server device when it opened its BluetoothServerSocket.

```
mmSocket.connect();
```

Second, we initiate the connection by calling connect().

Upon this call, the system will perform an SDP lookup on the remote device in order to match the UUID. If the lookup is successful and the remote device accepts the connection, it will share the RFCOMM channel to use during the connection and connect() will return. This method is a blocking call. If, for any reason, the connection fails or the connect() method times out (after about 12 seconds), then it will throw an exception.

Because connect() is a blocking call, this connection procedure should always be performed in a thread separate from the main activity thread.

```
mBluetoothAdapter.cancelDiscovery();
```

cancelDiscovery() is always called before the connection is made. We should always do this before connecting and it is safe to call without actually checking whether it is running or not.

```
mmSocket.close();
```

When we are done with our BluetoothSocket, we have to call close() to clean up. Doing this will immediately close the connected socket and clean up all internal resources.

When we have successfully connected device. Each connected device, if there are more than one, will have a connected BluetoothSocket. This is state we can share data between devices. Using the BluetoothSocket, the general procedure to transfer arbitrary data is simple:

1. Get the InputStream and OutputStream that handle transmissions through the socket, via getInputStream() and getOutputStream(), respectively.

2. Read and write data to the streams with read(byte[]) and write(byte[])

```
bytes = mmInStream.read(buffer);
```

There are, of course, implementation details to consider. First and foremost, we should use a dedicated thread for all stream reading and writing. This is important because both read(byte[]) and write(byte[]) methods are blocking calls. Because in our application, we only have an InputStream, we are using only read(byte[]). read(byte[]) will block until there is something to read from the stream. So, we are having a loop in the thread which is dedicated to read from the InputStream.

```
mHandler.obtainMessage(MESSAGE_RECEIVED, bytes, -1, buffer)
.sendToTarget();
```

The constructor acquires the necessary streams and once executed, the thread will wait for data to come through the InputStream. When read(byte[]) returns with bytes from the stream, the data is sent to the main activity using a member Handler from the parent class. Then it goes back and waits for more bytes from

the stream.

```
public void cancel() {
        try {
                mmSocket.close();
        } catch (IOException e) {}
}
```

The thread's cancel() method is important so that the connection can be terminated at any time by closing the BluetoothSocket. This should always be called when you're done using the Bluetooth connection.

Chapter 4

Conclusion

The purpose of our thesis was to develop an embedded system for access control, which is based on RFID and is controlled by the Android application. Every system for access control needs credentials for the identification and a reader for reading those credentials. In our embedded system the RFID tags are used as credentials, and the RFID antenna is used as a reader. Despite the fact that there are a lot of different access control systems using RFID reader, we did not find an access control system that is controlled by an Android application. In our system when a specific user is recognized, the user's name is written on a specific screen, which is a part of the embedded system. At the same time, the name is sent via Bluetooth module, to the Android application on our mobile device. With that the Android application gains control over the system for access control.

There are many additions that could be added to our embedded system and to our Android application, which will improve them. The first feature that could be added is a two-stage authentication. For example, when the embedded access control system reads the card number with a RFID card reader and if the user is found, to show the keypad on the TFT touch screen or to show a password field in the Android application. With that the users can log in. The next improvements can be making a database for recording the users, adding or deleting a user, while a server program on a personal computer executes control and surveillance.

Bibliography

[1] Wikipedia available on: http://enẇikipediaȯrg/wiki/Access_control [Accessed: 31.08.2014]

[2] "The history of Bluetooth technology", *Shenzhen Bolutek Electronic Technology Co.,Ltd.*, 2012

[3] Michael Foley, "How does Bluetooth work? ", *Scientific American*, 2007

[4] Wikipedia available on: http://en.wikipedia.org/wiki/Bluetooth [Accessed: 29.07.2014]

[5] By Mark Roberti , "The History of RFID Technology", *RFID Journal*, 2005.

[6] By Linda Castro, Samuel Fosso Wamba , "AN INSIDE LOOK AT RFID TECHNOLOGY ", *Journal of technology management and innovation*, vol. 2, issue 1, 2007.

[7] Yaara Lancet (2012) available on: http://www.makeuseof.com/tag/differences-capacitive-resistive-touchscreens-si/

[8] Ankur Tomar (2012) available on: http://www.element14.com/community/docs/DOC-42374/l/coocox-development-tool-for-arm-cortex-m-based microcontrollers#anchor1

[9] Wikipedia available on: http://en.wikipedia.org/wiki/Android_(operating_system) [Accessed: 11.08.2014]

[10] STM32F405xx/STM32F407xx Datasheet available on: http://www.st.com/web/en/resource/technical/document/datasheet/DM00037051.pdf [Accessed: 30.07.2014]

[11] STM32F405xx/STM32F407xx Reference manual available on: http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf [Accessed: 30.07.2014]

[12] HC Serial Bluetooth Products User Instructional Manual available on: http://www.exp-tech.de/service/datasheet/HC-Serial-Bluetooth-Products.pdf [Accessed: 04.08.2014]

[13] Samsung I9105 Galaxy S II Plus Specifications available on: http://www.gsmarena.com/samsung_i9105_galaxy_s_ii_plus-5213.php [Accessed: 12.08.2014]

[14] Bluetooth available on: http://developer.android.com/guide/topics/connectivity/bluetooth.html [Accessed: 03.05.2014]

[15] Richard Barry, "Using the FreeRTOS Real Time Kernel, Arm Cortex-M3 Edition", 2010

[16] Get the Android SDK available on: http://developer.android.com/sdk/index.html [Accessed: 01.09.2014]